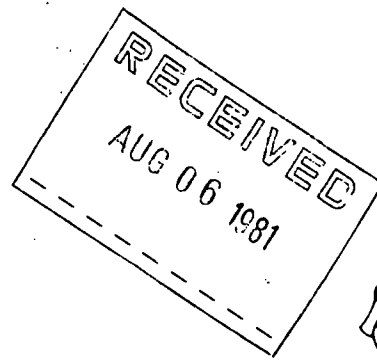


DAN 2426

RADC-TR-81-105

Final Technical Report

June 1981



SPECIFICATION TOOLS ENVIRONMENT STUDY

TRW Defense and Space Systems Group

L. Baker

M. R. Nixon

J. T. Lawson

D. A. Richard

R. P. Loshbough

R. A. Vossler

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

**ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441**

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

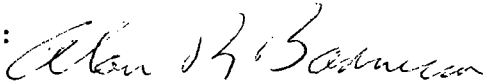
RADC-TR-81-105 has been reviewed and is approved for publication.

APPROVED:



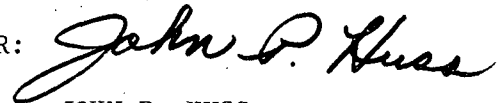
WILLIAM E. RZEPKA
Project Engineer

APPROVED:



ALAN R. BARNUM
Assistant Chief
Information Sciences Division

FOR THE COMMANDER:



JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

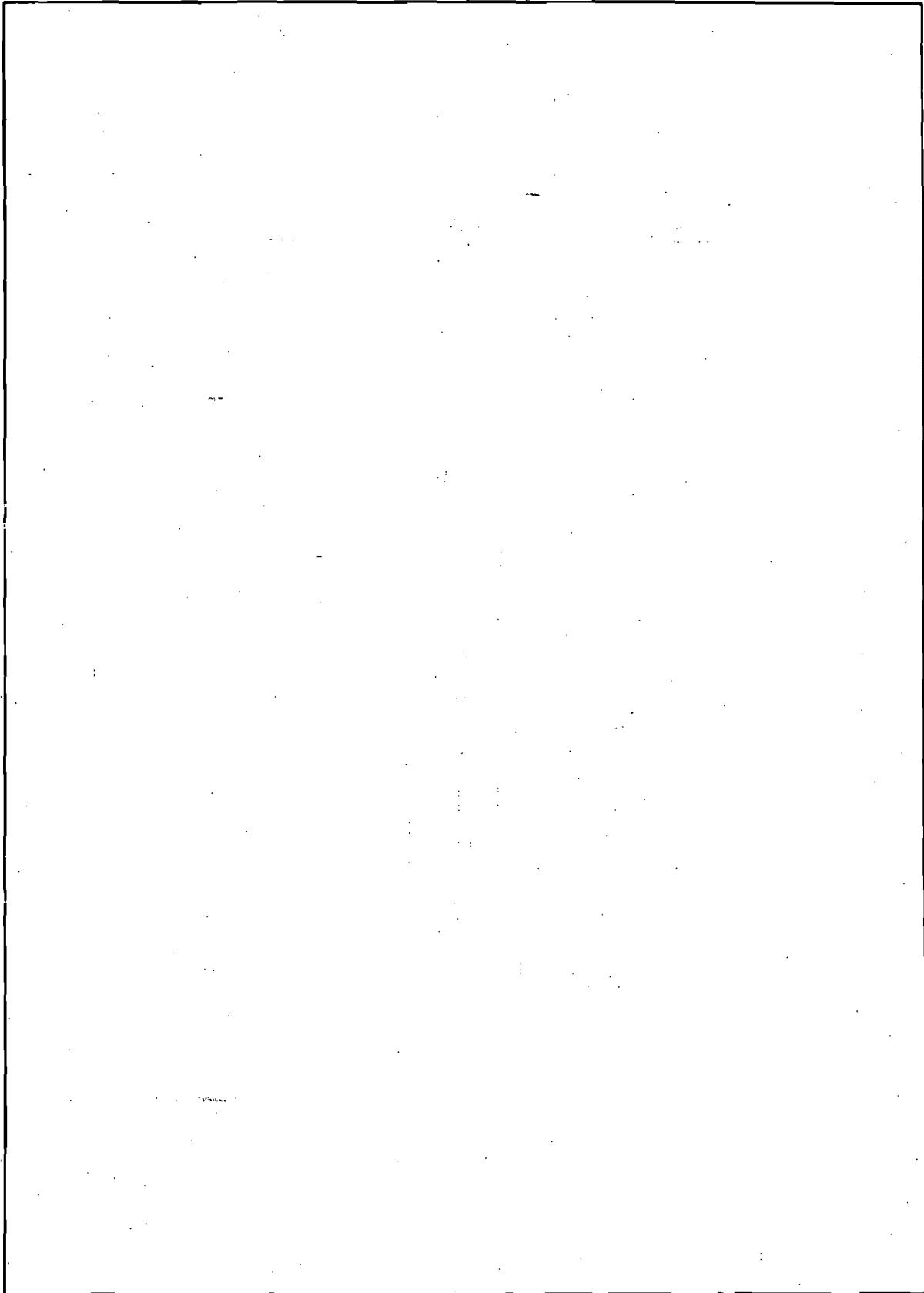
REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS. BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-105	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SPECIFICATION TOOLS ENVIRONMENT STUDY DAN 2426		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Nov 79 - Dec 80
7. AUTHOR(s) L. Baker, J. T. Lawson, R. P. Loshbough, M. R. Nixon, D. A. Richard, R. A. Vossler		6. PERFORMING ORG. REPORT NUMBER 35983-6921-002
9. PERFORMING ORGANIZATION NAME AND ADDRESS TRW Defense and Space Systems Group 7702 Governors Drive West Huntsville AL 35805		8. CONTRACT OR GRANT NUMBER(s) F30602-80-C-0026
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55812201
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		12. REPORT DATE June 1981
		13. NUMBER OF PAGES 158
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release, distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: William E. Rzepka (ISIE)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Specification Tools ARPANET Hosts REVS PERCAM AUTOIDEF		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report documents results of a study to determine an appropriate hardware/software/communication environment for three specification tools: REVS, PERCAM, and AUTOIDEF. Host systems currently on the ARPANET and alternate high performance midcomputers were evaluated against functional requirements for the STE. The feasibility of converting REVS, now coded in Pascal, into ADA, JOVIAL J73, FORTRAN or COBOL was also assessed.		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
1.0	EXECUTIVE SUMMARY	1
1.1	INTRODUCTION AND BACKGROUND.	1
1.2	OBJECTIVES AND SCOPE	2
1.3	STUDY CONCLUSIONS AND RECOMMENDATIONS.	3
1.4	ORGANIZATION OF THIS REPORT.	4
2.0	THE TOOLS AND THEIR SOFTWARE ENVIRONMENT.	5
2.1	METHODS OF INVESTIGATION: TASK 1.	5
2.2	RSL/REVS	6
2.2.1	Requirements Statement Language (RSL)	6
2.2.2	Requirements Engineering and Validation System (REVS)	7
2.2.3	Major Components of REVS and Its Support Software.	11
2.2.4	REVS Job Control.	14
2.2.5	Current REVS Installations.	25
2.2.6	REVS Software Environment Requirements.	25
2.3	PERFORMANCE AND CONFIGURATION ANALYSIS MODEL (PERCAM).	32
2.3.1	Event Logic Trees	35
2.3.2	PERCAM Software	37
2.3.3	Current Installations	42
2.3.4	PERCAM Software Environment Requirements.	42
2.4	AUTOIDEF*.	43
2.4.1	AUTOIDEF Software	44
2.4.2	Current Installations	49
2.4.3	AUTOIDEF Software Environment Requirements.	49
2.5	GENERAL SOFTWARE ENVIRONMENT REQUIREMENTS.	51
2.5.1	Source Library Control Requirements	51
2.5.2	Data Entry Requirements	52
2.5.3	On-Line Response Time Support	52
2.5.4	Communications Support.	52
3.0	HARDWARE RESOURCE REQUIREMENTS.	53
3.1	REVS RESOURCE REQUIREMENTS	53
3.1.1	CPU Speed	53
3.1.2	Primary Storage	54
3.1.3	Mass Storage.	55
3.1.4	REVS Interactive Color Graphics Requirements.	58
3.1.5	Plotter Hardware.	59
3.1.6	Card Reader/Alphanumeric CRT Terminals.	59

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
	3.1.7 Tape Unit	60
	3.1.8 Line Printer.	60
	3.1.9 External Communications	60
	3.1.10 Internal I/O Transfer	60
3.2	PERCAM RESOURCE REQUIREMENTS	60
	3.2.1 CPU Speed	60
	3.2.2 Primary Memory.	61
	3.2.3 Mass Storage.	61
	3.2.4 Graphics Hardware	61
	3.2.5 Plotter Hardware.	61
	3.2.6 Card Reader/Tape Unit/Line Printer.	61
3.3	AUTOIDEF RESOURCE REQUIREMENTS	61
	3.3.1 CPU Speed	61
	3.3.2 Primary Memory.	62
	3.3.3 Mass Storage.	62
	3.3.4 Graphics Hardware	62
	3.3.5 Plotter Hardware.	63
	3.3.6 Card Reader/Tape Unit/Line Printer.	63
3.4	STE HARDWARE SELECTION CONSIDERATIONS.	63
4.0	TOOL UTILIZATION PROFILES	65
	4.1 REVS UTILIZATION	65
	4.2 PERCAM UTILIZATION	71
	4.3 AUTOIDEF UTILIZATION	72
5.0	STE REQUIREMENTS SUMMARY.	73
	5.1 SOFTWARE ENVIRONMENT REQUIREMENTS SUMMARY.	73
	5.2 HARDWARE ENVIRONMENT SUMMARY	74
	5.3 STE INTEGRATION ISSUES	75
6.0	SURVEY OF ARPANET SYSTEM HOSTS.	76
	6.1 METHODS OF INVESTIGATION: TASK 2.	77
	6.2 CDC ARPANET HOSTS.	77
	6.2.1 Hardware/Software Functional Requirements	77
	6.2.2 CDC Host Job Scheduling Considerations.	79
	6.2.3 CDC Host Utilization Considerations	83
	6.2.4 CDC Host Job Billing Considerations	84

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
6.3	NON-CDC ARPANET HOSTS PARTIALLY EVALUATED BY TRW . . .	88
6.3.1	Hardware/Software Functional Requirements . . .	88
6.3.2	Non-CDC Host Processor Slowdown Ratios.	92
6.3.3	Host Utilization Considerations	92
6.3.4	Host Job Scheduling and Billing Considerations. . .	93
6.3.5	Job Scheduling Parameter Considerations at ANL. . .	97
6.4	HONEYWELL 6180 HOSTS	100
6.5	REVS RUN COSTS AT NADC	100
6.6	ARPANET HOST CONCLUSIONS	102
7.0	SURVEY OF ALTERNATE STE HOSTS	103
7.1	METHODS OF INVESTIGATION: TASK 3.	103
7.2	COMPARATIVE DATA ON ALTERNATE STE HOST MACHINES. . .	104
7.3	MIDICOMPUTER HOST SELECTION.	106
7.4	STE COLOR GRAPHICS TERMINALS	107
8.0	STE SECURITY CONSIDERATIONS	109
8.1	GENERAL SECURITY CONSIDERATIONS.	109
8.2	STE IMPLICATIONS	110
9.0	REVS IMPLEMENTATION LANGUAGE CONVERSION EVALUATION.	111
9.1	CANDIDATE LANGUAGE SUMMARY	111
9.2	CONVERSION IMPACT UPON EXISTING REVS FEATURES.	113
9.2.1	REVS Program Architectures.	113
9.2.2	Dynamic Storage Management.	114
9.2.3	Recursive Subprogram Calls.	116
9.2.4	Structured Programming Techniques	117
9.2.5	Data Structures	118
9.2.6	Automatic Consistency Checking.	119
9.2.7	Input/Output.	119
9.2.8	Automatic Generation of RSL Translator.	119
9.2.9	Automatic Generation of Simulator Program	121
9.2.10	Automatic Generation of Simulation Post-Processor	124
9.3	CONVERSION COST/SCHEDULE	125
9.4	CONVERSION IMPACT UPON LIFE-CYCLE MAINTENANCE.	127
9.5	LANGUAGE CONVERSION CONCLUSIONS.	128
10.0	REFERENCES:	129
APPENDIX A.	130

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
2-1	REVS Functional Organization.	12
2-2	REVS Job Sequence	19
2-3	PERCAM Multi-Simulator Philosophy	33
2-4	The PERCAM Analysis Process	34
2-5	ELT Example	36
2-6	User and Simulator Data Input Summary	41
2-7	Example IDEF Diagram.	45
2-8	AUTOIDEF System Overview.	46
4-1	Segment of REVS Project Run History	67
4-2	REVS Runs/Day Frequencies	68
4-3	Intervals Between REVS Runs	69
4-4	Time of Day of REVS Runs.	70
9-1	Simulator Program Overview.	122

LIST OF TABLES

<u>Table</u>	<u>Title</u>	<u>Page</u>
2.1	Current REVS Installations.	26
3.1	Installation Phase Mass Storage Files for REVS Software . . .	56
3.2	Production Phase Mass Storage Files for REVS Software	57
4.1	REVS Installation Runs.	66
4.2	Example PERCAM Study Number One	71
6.1	CDC ARPANET Host Evaluation	78
6.2	Input Queue Classes (C Parameter)	81
6.3	Age Computation for Input Queue (A Parameter)	81
6.4	PERCAM Study Number One Cost Estimates (LBL).	89
6.5	PERCAM Study Number Two Cost Estimates (LBL).	90
6.6	Non-CDC ARPANET Host Evaluation	91
6.7	PERCAM Study Number One Cost Estimates (ANL).	98
6.8	PERCAM Study Number Two Cost Estimates (ANL).	99
6.9	NADC REVS Costs	101
7.1	Candidate Midicomputer Hosts.	104
9.1	Candidate Language Comparisons.	126

1.0 EXECUTIVE SUMMARY

1.1 INTRODUCTION AND BACKGROUND

The production of high-quality specifications for systems and software is a long-standing and well recognized problem area. Traditional forms of specification, usually in free-form English text, have persistently been plagued with deficiencies such as incompleteness, inconsistency, ambiguity and other errors. Verification of such specifications is usually manual, hence, itself error-prone and often incomplete.

To address aspects of this problem area, several promising automated tools have been developed in recent years. Three particular tools of interest to RADC are RSL/REVS [1,2], PERCAM [3,4], and AUTOIDEF [5,6,7]. A summary description of each tool follows:

- RSL/REVS. The Requirements Statement Language (RSL) and the Requirements Engineering and Validation System (REVS) are components of the Software Requirements Engineering Methodology (SREM) developed by TRW for the U.S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC). The REVS software provides capabilities to translate RSL requirements statements, maintain a requirements data base, analyze the relational data base for desirable properties or errors, extract particular subsets of the data base under user control, and generate executable simulations constrained by the requirements statements. REVS consists of a set of Pascal programs, a Data Base Control System written in FORTRAN, and a set of support utilities including the TRW Pascal Compiler Writing System.
- PERCAM. This tool, developed by TRW for the rapid construction and execution of modular simulations, was originally used in the simulation of large-scale air defense attacker/defender scenarios. Subsequently, PERCAM has been used in preliminary performance studies for a number of different systems where resource constraints are a significant issue.
- AUTOIDEF. This large FORTRAN software package is being developed by Boeing Computer Services under contract to SofTech, Inc., for support of the Air Force ICAM program. This tool automates IDEF diagrams, a specialized form of the diagrams used by SofTech in their Structured Analysis and Design Technique (SADT). Build 1, which became operational on 10 August 1979, provides the capability to create and store IDEF diagrams. Build 2, yet to be completed, provides consistency checking capability.

These tools were developed at different times, for different purposes on various types of Control Data Corporation (CDC) computers. In order to further evaluate, improve and integrate these tools, and eventually make the products widely available within the Air Force, a compatible hardware/software/communication environment is needed. This environment has been named the Specification Tools Environment (STE). The study reported in these pages is concerned with the requirements upon the STE and candidate ways of implementing those requirements.

1.2 OBJECTIVES AND SCOPE

The objective of the STE Study was to determine hardware, software, and communication configurations for implementation of an environment capable of supporting the integration and development of automated tools for the analysis of system and software specifications. Existing environment implementations by industry and Government sources as well as new proposed implementations were considered.

The objective was accomplished through four study tasks as follows:

- Task 1: Define STE Functional Requirements
- Task 2: Determine Adequate ARPANET Hosts
- Task 3: Survey Alternate STE Hosts
- Task 4: Evaluate REVS Language Conversion.

The first task was aimed at determining the hardware, software, and communication requirements imposed on the environment by the tools. The second task evaluated the suitability of existing ARPANET systems using the requirements developed in the first task. The third task evaluated standard computer industry hardware/software configurations for possible use in a dedicated environment. The fourth task assessed the feasibility and approximate cost to rewrite the Requirements Statement Language/Requirements Engineering Validation System (RSL/REVS) into some other more widely available language.

The last task was included because RSL/REVS is written in the Pascal language, not one of the approved Air Force standard higher-order languages. Not only is REVS in Pascal, it fully utilizes all of the features of the language. When the STE Study was initiated in 1979, few available Pascal compilers fully implemented the language. In the past year the situation has dramatically improved as a number of powerful Pascal compilers have been introduced on a wide range of machines.

1.3 STUDY CONCLUSIONS AND RECOMMENDATIONS

There are three nodes on the ARPANET using CDC computer systems that have immediate capability to provide the STE. These are Brookhaven National Labs (BNL), Lawrence Berkeley Labs (LBL), and the Naval Air Development Center (NADC). Two of the STE tools, REVS and PERCAM, are already installed at NADC. Installation of REVS at one of the other nodes would require one-half to two man-years of technical effort, depending upon details of the host operating system. AUTOIDEF would have to be installed at any of the nodes.

Of the non-CDC ARPANET nodes, Argonne National Labs (ANL) and University of California at Los Angeles (UCLA-CCN), provide the best capabilities. Both nodes have large IBM 3033 mainframes. Using the new IBM Pascal compiler, it is believed that REVS could be installed on an IBM machine with about two man-years of technical effort.

Two Univac systems on the ARPANET have the processing power to provide the STE, but there are still deficiencies in the features of existing Pascal compilers for Univac machines. A DEC VAX 11/780 at the Naval Underwater Systems Center (NUSC) could provide the STE if main memory was upgraded to two megabytes from the present one megabyte. (REVS is being installed on the VAX under contract to the Ballistic Missile Defense Advanced Technology Center (BMDATC) in Huntsville, Alabama.)

Cost and availability information for ARPANET nodes was fragmentary and difficult to acquire. While sufficient information was provided by ANL and LBL to derive cost estimates for REVS runs, and these estimates were comparable with each other, they were significantly lower than results derived from a cost estimating relationship based on actual REVS runs at NADC. Comparative benchmark runs at candidate nodes would be necessary to verify real costs. There are three potential disadvantages of using an existing ARPANET node to host the STE: 1) possible unavailability of the node when needed; 2) inability to tailor the node configuration to STE needs; and 3) inability to process classified data without expensive add-on equipment.

To promote widespread technology transfer of STE tools throughout the Air Force, there is great merit to hosting the tools on a popular machine that is affordable, in terms of cost, to a wide spectrum of users. Our survey of alternate hosts indicates that the DEC VAX 11/780 is currently the best candidate to further this objective. Our survey of color graphics terminals resulted in selection of the Tektronix 4027 terminal for STE support.

We assessed the effort required to convert REVS to one of the standard languages, JOVIAL J73, FORTRAN, or COBOL, or to the forthcoming DoD standard language, ADA. We concluded that conversion to ADA would provide the greatest future benefit at least cost, followed by JOVIAL J73 in second place. Conversion to FORTRAN or COBOL would be costly, and since these languages are becoming obsolescent in the future, such a conversion would be of minimal benefit. With the recent introduction of a number of good Pascal compilers and the exploding popularity of the language, we recommend that the Pascal implementation of REVS be maintained until sufficient reliable ADA compilers are available and the future of ADA and JOVIAL J73 becomes more clear.

1.4 ORGANIZATION OF THIS REPORT

The remainder of this report documents methods of investigation and specific results for the four tasks of the STE Study. Task 1 results are reported in Sections 2 to 5. Task 2 results are reported in Section 6. Sections 7 and 8 address the results of Task 3, while Section 9 is devoted to Task 4. Section 10 lists major references cited in the report.

Section 2 is devoted to descriptions of the tools and their software environment requirements. Each of these tools is addressed separately. Paragraph 2.5 deals with utility software needed for the STE independent of the particular tools.

Section 3 is devoted to hardware resources necessary to support each tool. Where a tool has been developed to interface with specific hardware, we identify the particular item and its main characteristics. Discussion of substitute equipment and necessary tool modifications is inserted when appropriate. Quantitative estimates are also made for memory and storage sizing.

Section 4 presents run time and workload estimates for each tool based upon current installations and observed usage patterns. This information was used to evaluate speed requirements for various candidate configurations and operating policies. Section 5 concludes the results of Task 1 with a consolidated summary of requirements for the STE, considering the combined needs for REVS, PERCAM, and AUTOIDEF. Two integration issues are identified for future consideration.

Section 6 discusses results of a survey of candidate ARPANET hosts for the STE. Although personnel from several ARPANET nodes readily cooperated to provide their best information, we were surprised to find that reliable, current information is generally not available. Even the ARPANET Network Information Center (NIC) has not been able to accumulate accurate cost and workload data.

Section 7 summarizes the evaluation of alternate STE hosts of a type capable of near-term connection to the ARPANET and suitable for use in a dedicated STE mode. Since current ARPANET nodes include the common large mainframes provided by all of the major computer vendors, our emphasis was on investigating economical high performance "midcomputer" systems. Candidate color graphics terminals are also discussed.

Section 8 discusses the implications of operating the STE in a dedicated, single-level security mode. The discussion assumes use of a vendor-supplied commercial operating system.

Section 9 assesses the feasibility of converting the REVS tool, now coded in Pascal, to one of the approved DoD higher order languages: ADA, JOVIAL J73, FORTRAN, or COBOL. The more modern languages, ADA and J73, were found to be more cost-effective candidates than the early languages, FORTRAN and COBOL.

Section 10 lists references cited in the report, and Appendix A summarizes information collected about the individual ARPANET nodes discussed in Section 6.

2.0 THE TOOLS AND THEIR SOFTWARE ENVIRONMENT

This section, and the following Sections 3, 4, and 5, provide results of Task 1 of the STE Study. The objective of Task 1 was to define the hardware/software functional requirements imposed on the STE by the characteristics of the three candidate STE tools: REVS, PERCAM and AUTOIDEF. The results of Task 1 were used as inputs to Tasks 2, 3, and 4.

The first paragraph in this section describes the methods of investigation used to accomplish Task 1. The subsequent paragraphs in this section present descriptions of each tool, specific concepts implemented in the tool, and its major software components. After the introductory discussion about each tool, its specific software requirements upon the STE are presented, based upon current implementation and installations. The last paragraph in the section addresses utility support independent of the tools considered.

2.1 METHODS OF INVESTIGATION: TASK 1

Our approach in conducting this study was to investigate the requirements of the STE with respect to software, hardware and communications. For each of the tools, TRW first gathered information concerning the following:

- Major source modules
- Programming language(s)
- Job control mechanisms
- Source inputs (e.g., system data, test cases)
- Utilities.

Each of the above areas was assessed to establish the requirements each of the tools imposed on the STE.

Next we examined the equipment currently used to host each tool. The current installations and hardware used were originally defined either by 1) specific customer requirements, or 2) availability (the customer happened to have specific equipment, but this was not mandatory to support the tool). Regardless of the original requirement, the current installations are the reference point from which the transfer of the tools to a different environment must be assessed. Substantial departure from current features and assumptions may add significant tool modification costs to adapt to a new environment.

Third, we gathered current performance and workload data to approximate the STE load to be serviced. These will be used to assess various STE candidates. We have avoided stating explicit run time requirements for the tools because CPU speed alone is not a sufficient yardstick. Other factors such as availability of non-dedicated hosts, time-sharing service policy and cost/performance trade-offs will be major considerations in determining adequate configurations.

Finally, we examined the separate requirements for each tool, identified the most severe requirements over the set of tools, and synthesized a summary set of requirements for the STE as a whole.

2.2 RSL/REVS

The Requirements Statement Language (RSL) and the automated Requirements Engineering and Validation System (REVS) are two components of the Software Requirements Engineering Methodology (SREM) developed by TRW and delivered to the U.S. Army Ballistic Missile Advanced Technology Center (BMDATC) in 1976 as an increment in the BMDATC Software Development System (SDS). The third component of SREM, the methodology proper, directs the use of the language and tools in the development of software requirements.

As a starting point in this paragraph, we will describe RSL and REVS in sufficient detail to allow an understanding of the requirements upon the STE that follow. Because the REVS user interface has been designed for simplicity, it is difficult for the user to comprehend that what (to him) appears a monolithic program is, in reality, an intricate multi-job execution stream of a system of programs controlled by automated job stream manipulation within REVS. This process is discussed in detail in 2.2.4 and explains why transfer of REVS to a new environment is not a straight-forward task.

2.2.1 Requirements Statement Language (RSL)

RSL is a machine-readable, English-like language for stating software requirements. The basic structure of RSL is very simple and is based on four primitive language concepts: elements, attributes, relationships, and structures.

Elements

Elements in RSL correspond roughly to nouns in English. They are those objects and ideas which the requirements analyst uses as building blocks for his description of the system requirements. Each element has a unique name and belongs to one of a number of classes called element types. Some examples of standard element types in RSL are ALPHA (the class of functional processing steps), DATA (the class of conceptual pieces of data necessary in the system), and R_NET (the class of processing flow specifications).

Attributes

Attributes are modifiers of elements somewhat in the manner of adjectives in English; they formalize important properties of the elements. Each attribute has associated with it a set of values which may be mnemonic names, numbers, or text strings. Each particular element may have only one of these values for any attribute. An example of an attribute is INITIAL_VALUE which is applicable to elements of type DATA. It has values which specify what the initial value for the data item must be in the implemented software and for simulations.

Relationships

The relationship (or relation) in RSL may be compared with an English verb. More properly, it corresponds to the mathematical definition

of a binary relation, a statement of an association of some type between two elements. The RSL relationship is non-commutative; it has a subject element and an object element which are distinct. However, there exists a complementary relationship for each specified relationship which is the converse of that specified relationship. ALPHA INPUTS DATA is one of the relationships in RSL; the complementary relationship says that DATA is INPUT to an ALPHA.

Structures

The final RSL primitive is the structure, the RSL representation of the flow graph. Two distinct types of structures have been identified. The first is the R_NET (or SUBNET) structure. It identifies the flow through the functional processing steps (ALPHAs) and is thus used to specify the system response to various stimuli. The second structure type is the VALIDATION_PATH, which is used to specify performance of the system.

Through the use of these four primitive language concepts, a basic requirements language is provided which includes concepts for specifying processing flows, data processing actions, and timing and accuracy requirements. In addition, informative and descriptive material, and management-related information may be specified. The concepts of this language consist of twenty-one element types, twenty-one attributes, twenty-three relationships, and two types of structures. RSL can be extended to include additional concepts by defining new element types, attributes, or relationships. This allows the language to be tailored to the needs of a specific problem or project.

2.2.2 Requirements Engineering and Validation System (REVS)

The Requirements Engineering and Validation System (REVS) is an integrated system of software which aids in the development, maintenance, validation, and documentation of software requirements. REVS is designed to allow the requirements engineer to state and modify requirements information over a period of time as the requirements are developed. The RSL statements that an engineer inputs to REVS are analyzed, and a representation of the information is put into a centralized data base. This data base is called the Abstract System Semantic Model (ASSM) because it maintains information about the required data processing system (RSL semantics) in an abstract, relational model. Once entered into the ASSM, the requirements are available for subsequent refinement, extraction, and analysis by the REVS software.

From a user point of view there are five major functional capabilities which REVS provides:

- Processing of RSL.
- Interactive generation of Requirements Networks (R_NETs).
- Analysis of requirements and output of requirements in RSL and/or in specially formatted reports.

- Generation and execution of functional and analytic simulators from functional requirements and models or algorithms, and the generation and execution of simulation post-processors from analytic performance requirements.
- Processing of extensions to RSL.

REVS and RSL allow the engineer to enter requirements into REVS as they are developed, with REVS accumulating the information in the requirements data base and checking for consistency and completeness as new data is entered. Consequently, although the REVS capabilities may be applied in any order, in general, the user will initially enter RSL and then request various analyses to be performed by the RADX function. New entries will be made and analysis repeated until the requirements have been developed sufficiently for a simulation to be meaningful and useful. At that time, a simulator and post-processor may be generated. The simulator may then be executed numerous times and the data recorded and analyzed. Based on the results, this sequence may be repeated, starting with the modification of requirements already input to REVS or the addition of new ones. The sequence will also be repeated as system requirements change or new requirements are imposed. When the user is satisfied that the requirements are correct, based upon the results of static and dynamic analysis, REVS will provide outputs necessary to write a software requirements specification.

Each of the major capabilities identified above is allocated to a different functional component of REVS. The capabilities and the appropriate functions are described briefly below.

2.2.2.1 Processing RSL

The analysis of RSL statements and the establishment of entries in the ASSM corresponding to the meaning of the statements is performed by the RSL translation function of REVS. The translation function also processes the modifications and deletions from the data base commanded by RSL statements specifying changes to already-existing entries in the data base. For all types of input processing, the RSL translation function references the ASSM to do simple consistency checks on the input. This prevents disastrous errors, such as the introduction of an element with the same name as a previously-existing element, or an instance of a relationship which is tied to an illegal type of element. Besides providing a measure of protection for the data base, this type of checking catches (at an early stage) some of the simple types of inconsistencies that are often found in requirements specifications, without restricting the order in which the user adds to or alters the data base.

2.2.2.2 Interactive Generation of R-Nets

Graphics capabilities to interactively input, modify or display R_NET, SUBNET, and VALIDATION_PATH structures are provided through the REVS Interactive R-Net Generation (RNETGEN) function. RNETGEN permits entry of structures and referenced elements in a manner parallel to the RSL translator and thus provides an alternative to the RSL translator for the specification of the flow portion of the requirements. Using this function, the user may

develop (either automatically or under direct user control) a graphical representation of a structure previously entered in RSL. Through the use of the ASSM, the user may work with either the graphical or RSL language representation of a structure; they are completely interchangeable.

The Interactive R-Net Generation facility contains full editing capabilities. The user may input a new structure or he may modify one previously entered. At the conclusion of the editing session, the user may elect to replace the old structure with the modified one. The editing functions provide means to position, connect, and delete nodes, to move them, to disconnect them from other nodes and to enter or change their associated names and commentary. The size of a structure is not limited by the screen; zoom-in, zoom-out, and scroll functions are provided.

2.2.2.3 Analysis and Output of Requirements

The Requirements Analysis and Data Extraction (RADX) function provides both static flow analysis capabilities and the capabilities of a generalized extractor system to support both the checking for completeness and consistency in the requirements specification and the development of requirements documentation.

The static flow analysis deals with data flow through the R NETs. The analysis uses the R NET structure in much the same manner that data flow analyzers for programming languages use the control flow of the program to detect deficiencies in the flow of processing and data manipulation stated in the requirements.

The generalized extractor system allows the user to perform additional analysis and to extract information from the ASSM. The user can subset the elements in the ASSM based on some condition (or combination of conditions) and display the elements of the subset with any appended information he selects. By combining sets in various ways, he can detect the absence or presence of data, trace references on the structures, and analyze inter-relationships established in the ASSM. In analyzing user requests and extracting information from the ASSM, the extractor system uses the definition of the language concepts contained in the ASSM. Thus, as RSL is extended, the extensions and their use in the requirements are available for extraction.

2.2.2.4 Generation and Execution of Simulators and Post-Processors

The automatic Simulation Generation (SIMGEN) function in REVS takes the ASSM representation of the requirements of a data processing system and generates from it discrete event simulators of the DP system. These simulators are driven by externally generated stimuli (e.g., a weapon system model) known as a System Environment and Threat Simulation (SETS). This driver program models the threat, or situation, the system environment, and the components of a system external to the data processing system. Executable code for the SETS must be developed by the user, as are the executable code segments for ALPHAs described next

Two distinct types of simulators may be generated by REVS. The first uses functional (BETA) models of the processing steps and may employ simplifications to simulate the required processing. This type of simulation serves as a means to validate the overall required flow of processing against higher level system requirements.

The other type of simulator uses analytic (GAMMA) models (i.e., models that use algorithms similar to those which will appear in the software to perform complex computations). This type of simulation may be used to define a set of algorithms which have the desired accuracy and stability. Real-time feasibility of a system using this algorithm set is not established for any implementation; instead, the simulation provides an existence proof of an analytic solution to the problem. Both types of simulations are used to check dynamic system interactions.

Executable code segments representing the BETA and/or GAMMA modeling levels are developed by the user. These are stored in the ASSM as BETA and GAMMA text attributes of the corresponding processing step (ALPHA).

The SIMGEN function transforms the ASSM representation of the requirements into simulation code in the Pascal programming language. The flow structure of each R_NET is used to develop a Pascal procedure whose control flow implements that of the R_NET structure. Each ALPHA name on the R_NET is translated into a call to a procedure consisting of the model or algorithm (BETA or GAMMA) for the ALPHA. The models or algorithms are written in Pascal. The data definitions and structure for the simulation are synthesized from the requirements data elements and their relationships and attributes in the ASSM.

By automatically generating simulators in this manner from the ASSM, the simulations are insured to match and trace to the requirements. New simulators can be generated readily as requirements change; all changes are made to the requirements statements themselves, and are automatically reflected in the next generation of the simulator.

For analytic simulations, SIMGEN also generates simulation post-processors based on the statement of performance requirements in the ASSM. Data collected from an analytic simulation can be evaluated using the corresponding post-processor to test that the set of algorithms meet the required accuracies.

Both REVS generated simulators and post-processors are accessed for execution through REVS functions; the Simulation Execution (SIMXQT) function for simulators, and the Simulation Data Analysis (SIMDA) function for simulation post-processors.

2.2.2.5 Processing Extensions to RSL

An ASSM contains the RSL concepts used to express requirements as well as the requirements. Extensions and modifications to the concepts are processed by the RSL Extension translation (RSLXTND) function of REVS. The RSLXTND function is actually performed by the same software as RSL translation but is accessed separately to control extensions to the language through a lock mechanism built into the software.

2.2.2.6 REVS Organization

The above discussion has identified seven functions of REVS: RSL, RNETGEN, RADX, SIMGEN, SIMXQT, SIMDA and RSLXTND. As shown in Figure 2-1, these functions are under the control of a higher level function, the REVS Executive. The Executive presents a unified interface between the user and the different REVS functions.

2.2.3 Major Components of REVS and Its Support Software

Twelve major components, herein termed "source modules" are required for installation, execution, and maintenance of REVS. These vary in type from systems of programs to control decks and system data files transparent to the user.

2.2.3.1 Pascal Language Source Modules

- REVS: The Pascal application software itself consists of over 42,000 lines of code broken into more than 1100 procedures. To our knowledge, it is the largest Pascal program yet built.
- Compiler Writing System Package (CWS): This software package, adapted from a CWS developed by the University of Montreal, is used for generating the RSL translator within the REVS Pascal program. It consists of the following programs and segments:
 - SEMAGEN: This program pre-processes the extended Backus-Naur Form (BNF) of the RSL Language. As output, it generates tables for processing by other programs in the CWS.
 - LEXIGEN: This program produces the lexical analyses of the generated compiler.
 - COMPGEN: This program assembles all the generated compiler pieces into the compiler program.
 - SYNTGEN: This program builds the tables used by the standard syntax generator.
 - PROGGEN: This program generates the minimum programs from the syntactical structure of the input language.
 - NOYALEX: This is the Pascal skeleton for the lexical phase of the generated compiler.
 - NOYASYN: This is the Pascal skeleton for the syntactical phase for the generated compiler.
 - ERREURS: This is the Pascal skeleton for error recovery.
- Pascal Formatting Program: This program is used to generate consistent indentations, to re-position comment fields, and to assure that the Pascal code is limited to the first 72 print columns, in order to insure uniformity.

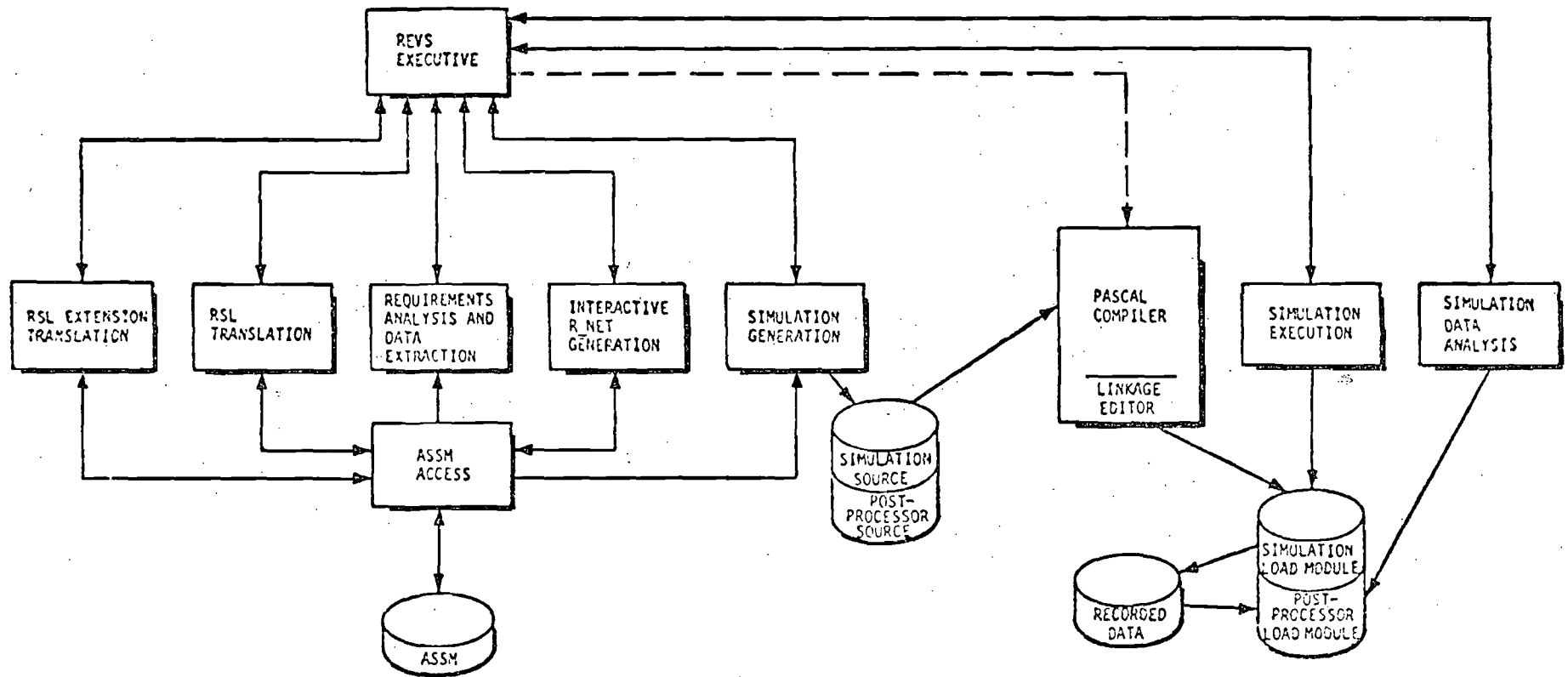


Figure 2-1 REVS Functional Organization

- RISF File: The Requirements-Independent Source File (RISF) contains Pascal code skeletons. The simulation generator of REVS uses this file to build the simulator program.

2.2.3.2 FORTTRAN Source Modules

- Data Base Control System (DBCS) Package: This software package, developed by the University of Michigan, contains the following source sub-modules:
 - The Data Base Control Systems (DBCS) Routines: These routines are the heart of the data base system. The DBCS routines actually handle the I/O for the data base.
 - The Data Definition Language Analysis (DDLA) Program: This program builds description tables and blocks the data areas required by the DBCS routines from the Data Definition Language (DDL) which describes the data base.
 - The DBCS Initialization (DBIN) Program: This program initializes a file for use in the data base.
 - DBCS Summary (DBSM) Program: This program produces a summary report as to the amount of data base in use.
- CALCOMP Compatible Plot Routines: These routines interface with the CALCOMP basic plot package to draw ellipses, circles, rectangles, and polygons at sites where the equivalent CALCOMP engineering drawing routines are not available.

2.2.3.3 Special Language Source Modules

These modules are special language inputs to the application software and transparent to the host system, or are job control and command instructions native to the host system.

- Data Definition Language (DDL) for the REVS RSL Data Base: This source module describes the structure of the REVS/RSL data base and is input to the DDLA program, described above.
- Data Definition Language (DDL) for the REVS Simulation Data Base: This source module describes the structure of the REVS Simulation data base and is used as input to the DDLA program.
- RSL Integrated Language Description for the CWS: This source module contains the information necessary for the CWS to generate the RSL translator. This is an extended BNF description.
- Segmented Overlay Commands (CDC-Dependent): These commands are input to the CDC Segmented Loader in order to describe the memory scheme to the loader.
- Job Control Language Decks for Building and Running REVS (CDC-Dependent): These control language decks are operating system and site-dependent.

- RSL Test Cases: These RSL input files are used to test the proper operation of REVS. Once REVS is installed, these tests can be used as a baseline for performance tuning.

2.2.4 REVS Job Control

REVS is invoked as a series of programs through the use of the job control statements of the operating system. This section illustrates job control on the CDC 7600, although it must be recognized that such control is dependent on the host computer operating systems. The following programs must be available at any host site:

- REVSPRE to initialize all files.
- REVSXQT to execute REVS.
- SIMBUIL to build the REVS-generated simulator and post-processor.
- SIMRUN to execute a REVS-generated simulator.
- TESTRUN to execute a REVS-generated post-processor.
- SIMSAVE to save a REVS-generated simulator and post-processor.
- SIMLOAD to reload a REVS-generated simulator and post-processor.

Following a description of these programs immediately below, their implementation via job stream control cards will be discussed.

2.2.4.1 REVSPRE Program

The REVSPRE program provides for the acquisition of necessary files and environmental conditioning for use of REVS on the CDC 7600. This program acquires all files needed to run REVS and is used only once at the beginning of a job deck. The files obtained are:

<u>LOCAL FILE NAME</u>	<u>PERMANENT FILE NAME</u>	<u>DESCRIPTION</u>
TAPE2	DBNUC	Predefined nucleus data base.
TAPE3	DBT	Data base tables.
TAPE10	VVDB	Empty post-processor data base.
TAPE11	VVDBT	Post-processor data base tables.
DONNEES	DONNEES	RSL translator input file.
RISF	RISF	SIMGEN input file used to construct a simulation program.
NESTER	NESTER	Utility program to reformat Pascal source programs.
PASCAL	PASCAL	Pascal compiler.

<u>LOCAL FILE NAME</u>	<u>PERMANENT FILE NAME</u>	<u>DESCRIPTION</u>
*REVSDB	DB	Empty data base.
*REVSABS	REVS200	REVS absolute with 200 pages in LCM.

All files are assumed to be disk resident on the account number PTCREVS. An alternate account number may be specified by invoking REVSPRE with that account number as a parameter, e.g., REVSPRE(acnt). REVSPRE will then attempt to acquire all files from this account and will note the account over-ride with a message in the job day-file.

The final operations performed by REVSPRE are the placing of a message in the day-file signifying REVSPRE completion.

2.2.4.2 REVSXQT Program

The REVSXQT program invokes execution of REVS and controls the acquisition and disposal of files needed during the REVS step. REVSXQT provides six positional parameters representing file names as follows:

REVSXQT(FILE01,FILE02,FILE03,FILE04,FILE05,FILE06).

The parameters and their default values are:

<u>FILE</u>	<u>DEFAULT</u>	<u>INTERPRETATION</u>
FILE01	INPUT	Standard REVS input file (REVS.IN).
FILE02	OUTPUT	Standard REVS log file (REVS.LOG).
FILE03	OUTPUT	Standard REVS output file (REVS.OUT).
FILE04	OUTPUT	DBCS TAPE6 error file (DBCS.ERR).
FILE05	OUTPUT	SIMGEN debug output file (REVS.DMP).
FILE06	PUNCH	Standard REVS punch file (REVS.PUN).

In addition to the six file over-ride parameters, two other parameters are provided. The seventh parameter is used to specify a local file name for a program to be executed instead of the nominal REVS absolute. A value of NO for the eighth parameter will inhibit the construction of a simulator and post-processor regardless of whether the REVS SIMGEN function is selected.

All files with the default name of OUTPUT will be automatically printed by REVS unless the file names are over-ridden on the REVSXQT card. The standard REVS punch file will also be automatically punched. If any of these file names are over-ridden on the REVSXQT call, the user assumes responsibility for their proper disposition.

*These files are acquired only to maintain their disk residence; they are immediately returned by REVSPRE.

The initial Phase 1 is conditioned by an empty REVSJSL file. In this phase, REVSXQT backs up the control card stream and then copies the users REVS inputs, either from INPUT or its over-ride, to the REVSIN file. REVSXQT then determines which REVS absolute program to execute, places a message in the job day-file, and makes a call to the loader to invoke that program. If a file name over-ride was given in the seventh parameter, this file name is used in the loader call. A message specifying the REVS absolute attached is put in the day-file and the loader is then called to execute this absolute.

Phase 2 occurs after the execution of the REVS program. In this phase, the Large Core Memory (LCM) scratch files used by REVS are returned, and the standard REVSLOG and REVSOUT files are copied to OUTPUT, or to user over-ride file names, preceded by banner pages giving the time, date, and file-id. Also copied out are the DBCS error file and the REVSDMP file if they are not empty. Likewise, a non-empty REVS punch file is copied to either PUNCH or an over-ride file, with no banner. Finally, the CALCOMP plot output file (TAPE4) is returned.

If no simulator was requested by SIMGEN or if it was suppressed by the eighth parameter, then REVSXQT terminates by returning the REVSJSL file, and putting a message in the day-file. Otherwise, the SIMBUIL program is invoked and the address at which to resume execution in REVSXQT is saved in the REVSJSL file. REVSXQT will terminate as above when control is returned by SIMBUIL.

2.2.4.3 SIMBUIL Program

The SIMBUIL program is called internally by REVSXQT to build a simulator and post-processor. SIMBUIL performs the following functions in order, each time incrementing the phase number and backing up the control card stream:

<u>PHASE NUMBER</u>	<u>ACTION</u>
2	Execute the NESTER program to reformat the generated simulator.
3	Execute the Pascal compiler on the nested simulator.
4	Construct and execute a file of control cards to link the simulator program.
5	Execute the NESTER program to reformat the generated post-processor.
6	Execute the Pascal compiler on the renested post-processor.
7	Construct and execute a file of control cards to link the post-processor program.

The result is a simulator program absolute on file REVSSIM and a post-processor absolute on file REVSVAL. Other files used in these phases either have been printed on output (compilation listings and load maps) or returned. SIMBUIL then returns control to REV SXQT through the return address saved in REVSJSL. Throughout the several phases of SIMBUIL, appropriate messages describing the actions being performed are placed in the job day-file.

2.2.4.4 SIMRUN Program

The SIMRUN program executes a simulator generated by the SIMGEN function of REVS. The simulator may have been generated in a previous REV SXQT step, or loaded from a tape or disk file by the SIMLOAD program, either of which will supply the associated files needed. Execution of the SIM SXQT function in a previous REV SXQT step is required to supply necessary simulator input. The SIMRUN program is invoked by a control card specifying the word SIMRUN.

On initial entry, SIMRUN will place a banner page on OUTPUT, set the phase number to eight, back up the job control card stream, and call the loader to execute the REVSSIM file. (Note: the banner page is written since REVSSIM may write directly on OUTPUT.)

After the simulator execution, any validation data generated is copied to OUTPUT preceded by a banner page, the phase number set to nine and the job control card stream backed up. The loader is called to execute the REVS post-processor data base builder (VVDBLDR). Following that execution, SIMRUN returns the REVSJSL file, and terminates with a message to the day-file.

2.2.4.5 TESTRUN Program

The TESTRUN program executes a simulation post-processor generated by the SIMGEN function of REVS. The post-processor may have been generated in a previous REV SXQT step, or loaded from a tape or disk file by the SIMLOAD program. The recording data base used by the post-processor is generated by execution of a simulator, and is saved along with a simulator by the SIMSAVE program. (A null data base is generated if the simulator is saved prior to execution in a SIMRUN step.) Execution of the SIMDA function in a previous REV SXQT step is required to supply necessary post-processor control input. The TESTRUN program is invoked by a control card specifying the word TESTRUN.

TESTRUN operates in a single phase and does not need to back up the job control card stream. First TESTRUN places a banner page on OUTPUT for the subsequent post-processor execution data. Then, the post-processor program (REVSVAL) is executed by a loader call. TESTRUN does not regain control after this execution.

2.2.4.6 SIMSAVE Program

The SIMSAVE program is used to combine REVS-generated simulator and post-processor related files onto a single file to assure their consistency. SIMSAVE copies the REVS generated simulator absolute (REVSSIM), the Event Enablement Definition File (EEDF), the REVS generated post-processor absolute (REVSVAL), and the post-processor data base file (TAPE10) onto a local file named SIMFILE.

These files are combined on a local file named SIMFILE. This file can then be saved on disk or tape by using the appropriate CDC control cards. The files combined on SIMFILE include the load modules for the simulator and post-processor programs, as well as the other files necessary for their execution. The recording data base generated by execution of a gamma simulator is included (a null data base is generated if the simulator is saved prior to execution in a SIMRUN step). The SIMSAVE program is invoked by a control card specifying the word SIMSAVE.

2.2.4.7 SIMLOAD Program

The SIMLOAD program reconstructs the REVS simulator and post-processor files previously saved by the SIMSAVE program. The SIMLOAD program assumes that these files are available on a local file named SIMFILE, the user is responsible for supplying the necessary CDC file cards to obtain this local file. After the SIMLOAD program is complete, the loaded simulator and post-processor may be executed by the SIMRUN and TESTRUN programs after the required SIMXQT and SIMDA inputs are supplied in a REV SXQT step.

The SIMLOAD program is used to retrieve the four files placed on SIMFILE by a previous execution of SIMSAVE. These files are copied from SIMFILE and placed on the files named REVSSIM, EEDF, REVSVL, and TAPE10.

2.2.4.8 REVS Control Stream Sequence

Using these programs, the normal job set-up can be very simple, requiring a job card to identify the job and acquire necessary resources. Even though the job set-up appears simple to the user, the actual job accomplishment depends on a complex job control stream sequence, which must be tailored for each host machine. Its implementation on the CDC 7600, partially described above, is presented in context in this paragraph.

REVS is executed within a multi-phase control stream. This control stream is necessary to control the different stages of processing that REVS must do. This section describes those phases and how they are controlled by the REVS program. Figure 2-2 illustrates the control stream and points (A through P) are cited throughout the following discussion.

REVS is initially called by the control statement called REVSPRE (A) which is brought into memory and generates the control statements for the file assignments in anticipation of the REVS run. After the file assignments are made, the user may use his own control statements to redefine the files that are different from the defaults. This allows the user to specify his own data base rather than the standard initial data base. When the REV SXQT statement is encountered (B), the REV SXQT control program finishes the file assignments required to run REVS, and generates the control statement to bring in the REVS program itself. After these control statements are processed (C) the REVS program executes and reads in the RSL, the REVS command language, and the RSL statements that are to be processed. The REVS program, at this point, consists of Pascal and the necessary data base to analyze RSL. Special communication is required between the REVS program and the REV SXQT program (D) where the program informs REV SXQT as to whether further processing is required for simulation.

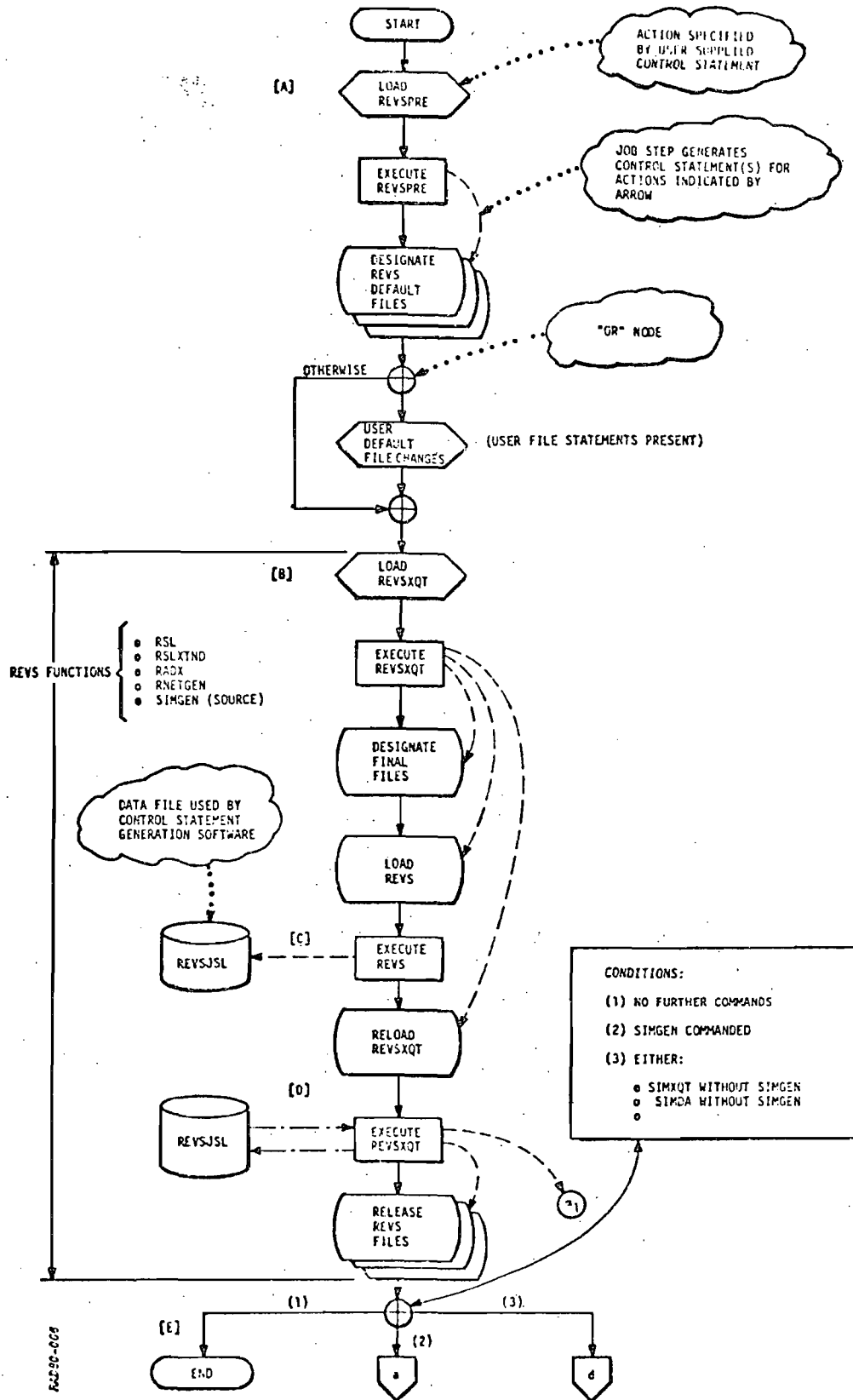


Figure 2-2 REVS Job Sequence

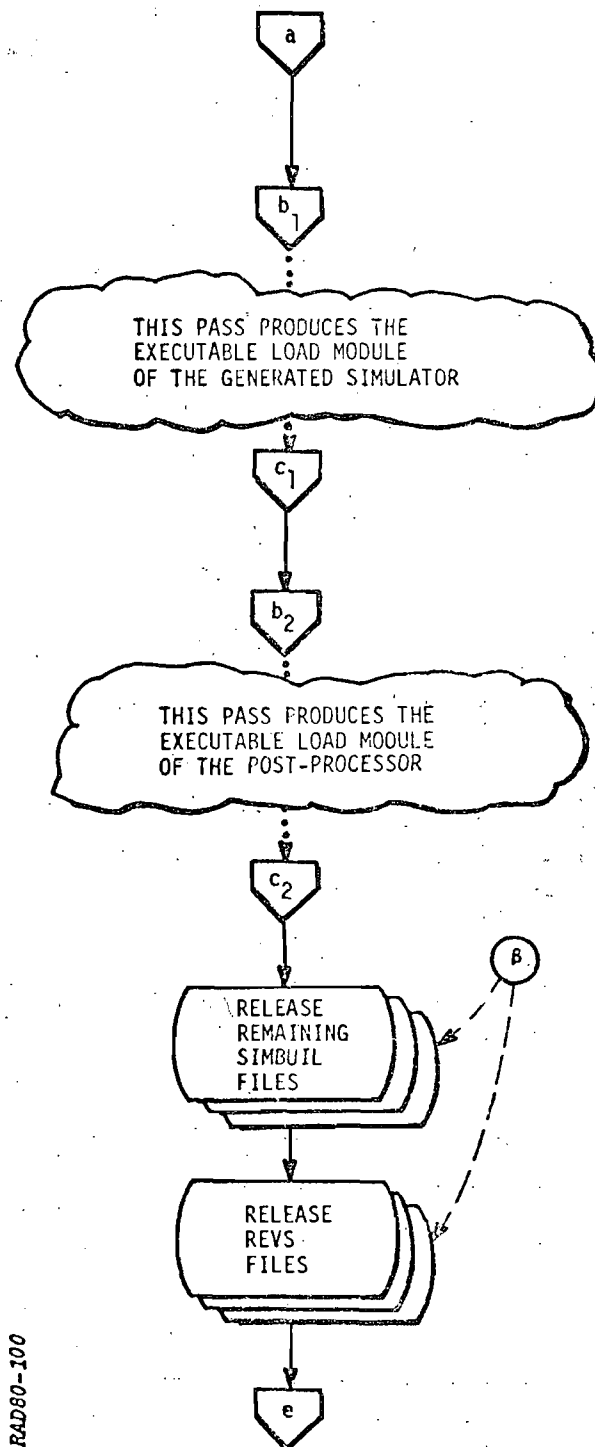


Figure 2-2 REVS Job Sequence (Continued)

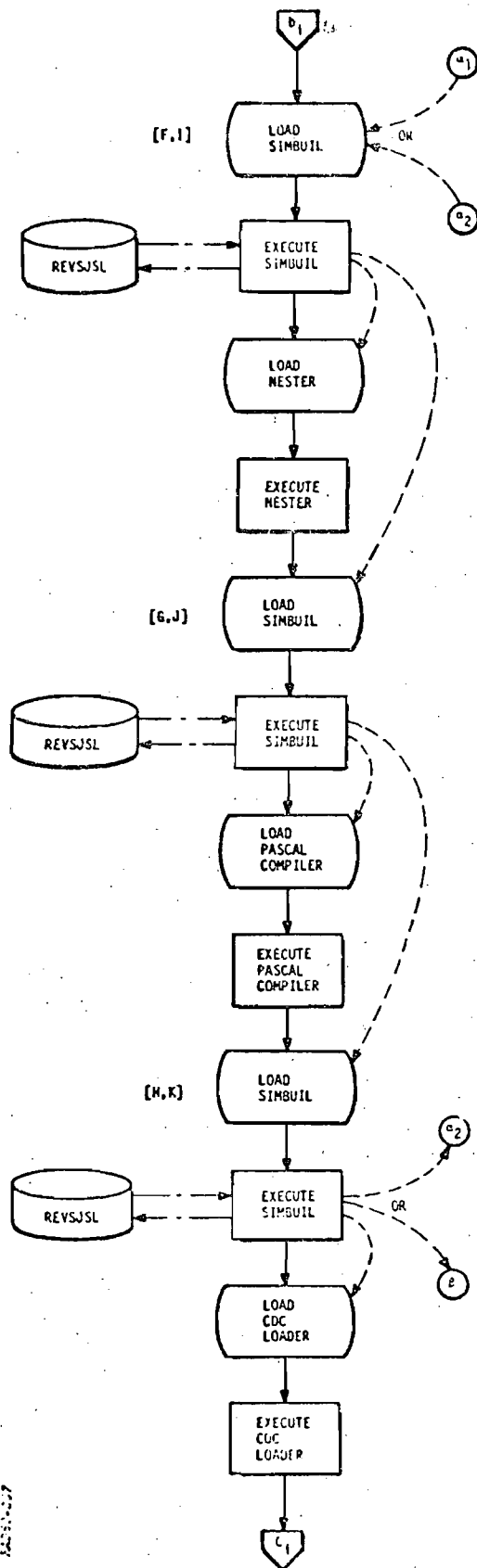


Figure 2-2 REVS Job Sequence (Continued)

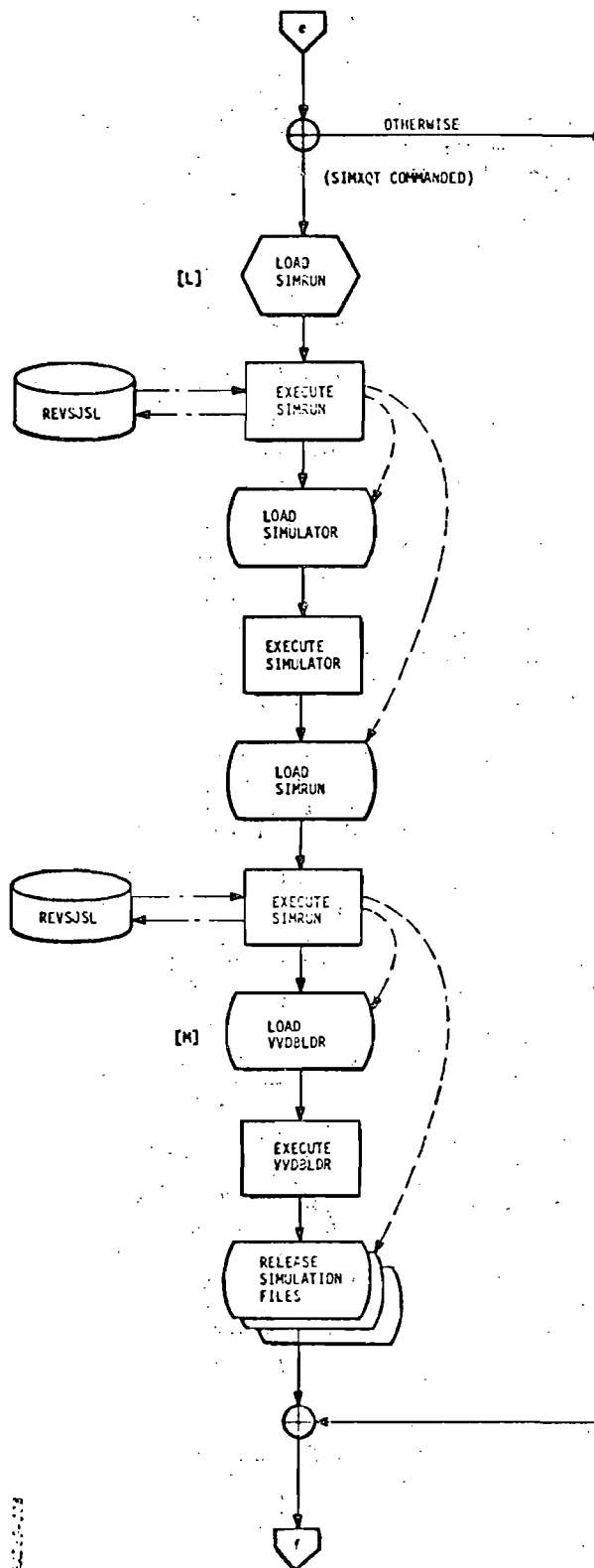


Figure 2-2 REVS Job Sequence (Continued)

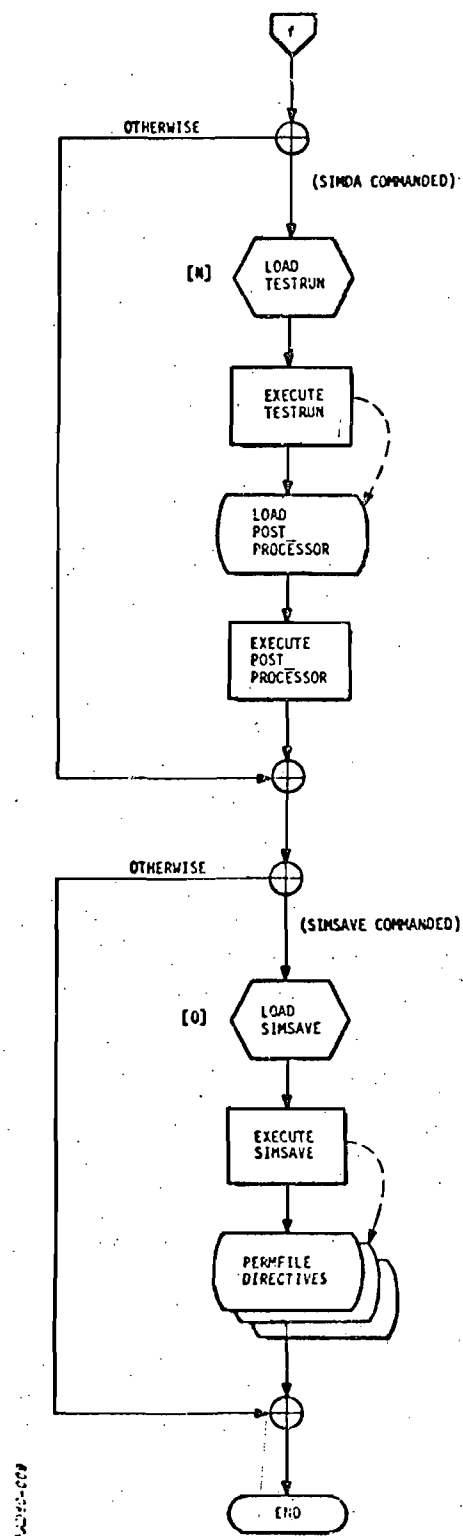
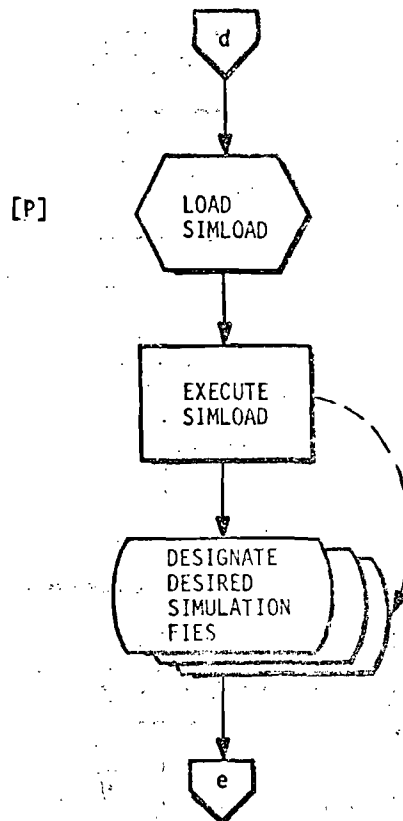


Figure 2-2 REVS Job Sequence (Continued)



RAD80-005

Figure 2-2 REVS Job Sequence (Concluded)

If simulation has not been requested, the REVS job stream terminates at this point (E). If a simulation is required, a control statement switches (F) to the beginning of the execution of the program SIMBUIL which constructs the simulator. First, the NESTER program is called to improve the format of the generated simulator. The SIMBUIL program generates the control statements to bring in SIMBUIL a second time (G), where control statements are generated by SIMBUIL to call the Pascal compiler to compile the generated simulation program. SIMBUIL is called in again (H) to generate control statements to bring in the loader to generate an absolute version of the simulation program. Next, SIMBUIL is brought in once more (I) to nest the Pascal generated post-processor. When that is completed, SIMBUIL is called again (J) to generate the statements to call the Pascal compiler to compile the post-processor. After that compilation, SIMBUIL calls the loader into memory to generate an absolute version of the simulator post-processor program (K). Then, at (L), a user-supplied control statement, SIMRUN, calls in the simulation run program. This program generates the necessary control statements to actually execute the simulator. The simulator is brought into memory, is executed, and then exits from memory. Control then brings in VVDBLDR, the data base builder program (M), executes it, builds a data base from the simulation run, and then exits from memory. At point (N), the user-supplied control statement TESTRUN may call the TESTRUN program to generate the control statements to run the post-processor. If the user wishes to save the simulator, the SIMSAVE command is issued by user which calls in the SIMSAVE program (O) to generate the necessary control statements to save the absolute of the simulator and its post-processor as a permanent file.

Once the simulator and post-processor have been saved, the sequence F to K need not be repeated for subsequent runs unless modifications are necessary. When the REVS command stream contains a SIMXQT or SIMDA command without a preceding SIMGEN command, REVSXQT at (D) will arrange for SIMLOAD to be brought in to recall the saved files and generate the proper control statements (P). The sequence L to O is then performed as before.

2.2.5 Current REVS Installations

REVS is currently operational at five sites shown in Table 2.1. The TI-ASC version at NRL was an early release and does not include subsequent changes and performance enhancements. In addition, this year, BMDATC is sponsoring transfer of REVS to a DEC VAX 11/780 at the Advanced Research Center (ARC).

2.2.6 REVS Software Environment Requirements

2.2.6.1 Pascal Compiler Requirements

The Pascal used in the REVS software is highly compatible with the "standard" set forth by Jensen and Wirth. However, there are some elements of the standard language which are explicitly stated to be implementation dependent. Other elements are defined in a manner which allows different interpretations. Still other elements which are clearly needed to support software development and maintenance are not part of the standard.

Table 2.1 Current REVS Installations

LOCATION	HOST	I/O CAPABILITIES			
		ON-LINE GRAPHICS	REMOTE BATCH	TELETYPE	OFF-LINE GRAPHICS
ADVANCED RESEARCH CENTER, HUNTSVILLE, ALABAMA	CDC 7600	X	X		X
NAVAL RESEARCH LABS, WASHINGTON, D. C.	TI-ASC			X	X
MDAC, HUNTINGTON BEACH, CALIFORNIA	CDC 7600		X		X
TRW, REDONDO BEACH, CALIFORNIA	CDC CYBER 74/174 TSS		X	X	X
NAVAL AIR DEVELOPMENT CENTER, WARMINSTER, PENNSYLVANIA	CDC CYBER 174 175		X X		X X

RAD80-020

TRW has spent considerable resources to modify and improve a basic CDC 6000 series Pascal system in order to implement a highly usable system capable of supporting REVS on several different CDC computing systems. At present, more compilers are being offered on the market by vendors of various machines. Not all of these provide the sophistication required to support REVS. An STE Pascal compiler must support the following REVS needs:

1. Nesting Limits - REVS is a highly structured software program. As such, there is extensive use of language construct nesting, both statically (e.g., procedures defined within procedures) and dynamically (e.g., recursive calls to handle recursive REVS data and control structures). The currently defined static nesting is nine levels deep while the dynamic nesting is a function of the RSL constructs and has no definite limit. Adaptation of REVS to nesting limits would be extremely difficult and would impose limits on the supported RSL constructs.
2. Procedure Size Limits - REVS contains some very large Pascal procedures, both in terms of the code space (~500 lines) and in terms of the number of variables defined (~300). Partitioning these procedures into smaller pieces would cause considerable breakage to the current REVS software structure.
3. Procedure Number Limits - The current REVS program contains over 1100 Pascal procedures and functions. Reducing this number would require a considerable effort to restructure the REVS code and duplicate blocks of code in several locations.
4. Dynamic Storage Implementation - REVS makes extensive use of the Pascal dynamic storage management to re-use the same storage area for multiple purposes. This requires a full implementation of both NEW and DISPOSE. Restructuring REVS to operate in a limited memory space without this implementation is probably not feasible. The alternative NEW, MARK, and RELEASE implementation is of only limited utility to REVS.
5. File Restrictions - REVS uses many Pascal files, both local to the REVS program (i.e., scratch files) and global (i.e., known outside the REVS program). These files are declared at several levels in REVS and are both textual and non-textual in nature. REVS could not be made operational without non-textual files. The inability to declare global files at any level would involve considerable breakage to the existing REVS software.
6. Interface to FORTRAN - The data base control system used to implement the REVS DBCS is written in FORTRAN and uses a FORTRAN random access file to maintain the data base. The Pascal system used must, therefore, support the calling of FORTRAN routines and functions from Pascal and must allow a mixture of FORTRAN and Pascal I/O (on different files).

7. Support of GOTO's - Several of the REVS functions use GOTO's which transfer control out of the procedure in which the GOTO appears. This is used to efficiently handle certain error conditions and to allow the termination of certain processes (e.g., user termination of on-line RADX output) without the need for extensive termination code. The revision of REVS to avoid the need for these GOTO's would be a substantial task and would increase the size of the REVS program.
8. Separate Compilation - It may be quite expensive to compile the entire REVS program every time a change is made. Frequent changes will undoubtedly be required during the REVS installation, with less frequent changes during the maintenance period. The inability to separately compile parts of REVS would be costly, probably both in time and computer resource usage. In addition, the current configuration of the REVS generated post-processor requires the existence of separately compiled Pascal procedures. The revision of the SIMGEN function to generate the necessary Pascal source for compilation each time is a substantial task.
9. Symbolic Debugging and Development Utilities - The ultimate life-cycle cost of any software product often depends on the quality of the debugging and program development utilities available. The cost-efficient modification, installation, and maintenance of the REVS software requires a highly developed set of such utilities. Required are user-callable symbolic stack and heap dumps, procedure call trace-backs, and extensive run-time error checking (e.g., bounds check, subscript checking). Also needed are stand-alone utilities such as a nester program to reformat the free-format Pascal to reflect the program structure and a cross-reference program to produce a concordance of program variable usage. It is expected that a significant part of the REVS installation effort may be devoted to enhancing existing utilities and implementing additional ones.
10. Character Set - The character set used by REVS consists of the upper case letters, digits, and a small number of special characters. No lower case letters or special control characters are used. The Pascal system, however, is assumed to allow the underscore and dollar sign characters in identifiers. Extensive use of the underscore is made in REVS generated simulators and post-processors. The dollar sign is used in the names of a few utility library routines called by REVS. If the dollar sign was not available the names of the routines would have to change, but this would only require selective recompilation. Lack of the underscore, however, would be unacceptable because the underscore is used as a connector to permit meaningful multi-word RSL element, attribute, and relation names.

Consider the following alternatives:

- With Underscore: DAY_OF_WEEK
- Without Underscore: DAYOFWEEK
- Short name, six characters as in FORTRAN: DYOFWK

By making identifier names look like words or sequences of words that are meaningful, the clutter of mnemonics and run-together words is lessened, thereby increasing the readability of the RSL. The hyphen could not be used, as in COBOL, because it is indistinguishable from the minus sign that must be used in REVS arithmetic operations, and would require analysis of the context in which it is used.

Contemplated extensions to REVS, such as the "pretty print" option will require the 96 character ASCII character set. Thus, it is desirable that both the compiler and the STE host support this character set.

11. External Files Processing - Standard Pascal recognizes both external and internal files. External files are those files that are listed in the program statement at the beginning of the program. These files can exist prior to and after the execution of the Pascal program. Internal files are files that are used during the execution and disappear when the Pascal program terminates. REVS requires all internal files to be treated as external files to retain them for later job steps, even though they do not appear in the name list of the program statement. Treating the internal files in this fashion is a special option in the current Pascal compiler. Absence of this option would require changing the source code to place all internal file names in the program statement.

2.2.6.2 Special Pascal/Operating System Communication Requirements

The Pascal environment must provide a means of communicating pertinent data to the O/S during the execution of the Pascal program. This communication includes such things as dynamically assigning files, determining time of day and date and switches to alter the job control stream.

REVS requires that files be dynamically opened and closed during the Pascal program execution. Standard Pascal allows externally-defined files to be identified only at the beginning of the Pascal program. This is done in the Pascal language by file names on the program statement. REVS Pascal dynamically assigns and deassigns files through two special routines. These routines are OPEN\$ and CLOSE\$. These routines associate a name with an external file, and are unique to the REVS Pascal environment. If the host system does not have this capacity, the means of attaching more than one add-file through the REVS control cards would not be possible. This would be a disadvantage in a "cardless" data entry system environment.

REVS Pascal informs the user of the time and date the program is run for documentation purposes. Standard Pascal does not have the time and date functions. These must be available for proper labeling of user listings.

During the execution of the REVS program, the user may specify several additional processing steps to be completed after its execution. These additional processing steps will require additional control statements. These are for R-Net plots and simulations. Although specific implementation is dependent upon the job control mechanism of the host system, the capability to modify the job control stream from within the Pascal program must be provided.

2.2.6.3 FORTRAN Compiler Requirements

REVS incorporates over 10,000 lines of FORTRAN code and interfaces with FORTRAN routines for CALCOMP plotting and interactive graphics. This code is ANSI FORTRAN 66 with the following exceptions, confined to the Data Base Control System (DBCS).

- Random File I/O - Random File I/O is required for file I/O to roll in and roll out data base pages. Since the data base file is not a sequential file, but random access, the FORTRAN execution environment must provide for random file I/O.
- Word Mask and Shift Operations - Word mask and shift operations are used to allow bit manipulation on data words of the host machine so as to allow the packing of multiple data items into one computer word. Using these extensions, it is possible to "tune" a host computer's data base control system through a trade-off between CPU, memory and I/O operations to attain an efficient REVS processing capability. This efficiency, however, can be attained only through handcrafting these modifications at each site. Through judicious tuning, the performance of the DBCS has been increased up to 100:1 over the original University of Michigan issue.

2.2.6.4 CALCOMP FORTRAN Plotting Requirements

The REVS system utilizes the basic CALCOMP FORTRAN plot routines. These routines are used to generate the R_NET plots from REVS. It is not necessary to provide CALCOMP software, but the routines must be functionally equivalent and present a calling interface identical to the CALCOMP routines: PLOTS, PLOT, NUMBER, and SYMBOL. The support software for Zeta Research plotters recently procured by RADC satisfies this requirement.

2.2.6.5 ARC Graphics System Software

At the BMDATC Advanced Research Center (ARC), the user application programs interface with the ARC Graphics System through a library of forty-one FORTRAN subroutines supporting seven functional groups:

- Console Initialization and Control - These subroutines permit access, control, and release of terminal sets (Group One).
- Keyboard Control and Data Interface - These subroutines cause the keyboards to be enabled or disabled and allow input of alphanumeric data from terminal keyboards (Group Two).
- Trackball Control and Data Interface - These subroutines cause the trackballs to be enabled or disabled and allow input of display cursor position from terminals (Group Three).
- Display Control - These subroutines communicate display screen position and color control information to the display generator (Group Four).
- Alphanumeric Information and Displays - These subroutines permit the drawing and formatting of characters and symbols on terminal display screens (Group Five).
- Graphic Displays - These subroutines are used to generate the data needed to draw figures and construct graphical presentations on terminal display screens (Group Six).
- Supporting Utility Capabilities - These subroutines are special-purpose capabilities that include interactive text editing, time delay control of application software, and video recording and recovery of screen displays (Group Seven).

REVS currently makes calls to 24 of these routines (two in Group One, five in Group Two, six in Group Three, two in Group Four, three in Group Five, five in Group Six, and one in Group Seven). These routines are specific to the ARC system, equipment, and operations philosophy (i.e., the software is not necessarily applicable outside the ARC environment).

The best strategy to follow for STE use would be to maintain the same REVS FORTRAN interfaces where relevant, and to implement new FORTRAN routines that are functionally equivalent, but which are adapted to the particular STE equipment, operational environment, and needs.

2.2.6.6 Memory Management Requirements

The CDC installation of REVS minimizes primary memory requirements through use of memory overlays, and is currently divided into some 40 overlays. The REVS overlay system requires that the overlay process be totally transparent to the source program. This means that nowhere within the source of REVS are there any calls to an overlay loader. The calling of code for an overlay must be entirely generated by the CDC segmented loader. To achieve the same results on a non-CDC system would require a similar overlay scheme. Any overlay scheme which required source code changes would be unacceptable. A virtual memory system is an alternative to overlays.

A virtual memory system is also an alternative to the current REVS method of paging in the data base from disk. Preliminary experiments on the VAX 11/780 indicate that the native virtual memory system is more efficient than paging via the DBCS. The memory management technique is more critical in limited core configurations, and will have to be assessed for each candidate STE configuration relative to its other parameters.

2.3 PERFORMANCE AND CONFIGURATION ANALYSIS MODEL (PERCAM)

PERCAM is a simulation system driven directly by Event Logic Tree (ELT) representations of systems. An ELT (see Paragraph 2.3.1) is a graph model representation which functionally represents a system's operational logic (sequence of events, time delays, and decision nodes). It was designed specifically to perform a range of analyses from quick turn-around cursory studies to in-depth analysis efforts. Within PERCAM, common system operations are stored as a set of components. Each component is a module of code (a Macro) that simulates a specific activity. Any ELT that is described in terms of these components can be processed by PERCAM to combine the component code into an executable module (the engagement model). If a system cannot be adequately described by the existing components, additional ones can easily be added. This capability frees the user from the programming details usually associated with simulation construction and allows the simulation to be constructed by the system analyst rather than an experienced programmer.

User-defined environmental conditions (e.g., scenario, threat) and system performance characteristics (e.g., sensor detection characteristics) can be input to an Engagement Model which executes in a Monte Carlo environment. A post-processor summarizes and tabulates results in a user-oriented format (tables, histograms, etc.).

PERCAM is currently being utilized to analyze a variety of system types (from tactical weapon systems to computer center operations) from a variety of standpoints -- from system performance effectiveness to computer loading and message traffic. Consequently, several component libraries exist for use depending on the nature of the problem to be addressed (see Figure 2-3). Currently, component libraries exist to analyze tactical systems from an effectiveness standpoint utilizing both the FORTRAN language as a base and the COMO simulation system as a base. Other component libraries exist to analyze system resource requirements utilizing the TRW developed SALSIM simulation language as a base.

The PERCAM analysis process, shown in Figure 2-4 begins with definition of a system in terms of ELT and analysis of key performance parameters. (Any components not adequately templated in the component library are also developed at this time.) Logic link specifications, corresponding to the ELTs, and performance data definitions are input to the Preprocessor (macro-processor simulator builder) which selects and conditions components from the library according to the user input. Specific operational data values are defined to the simulation executive at run-time.

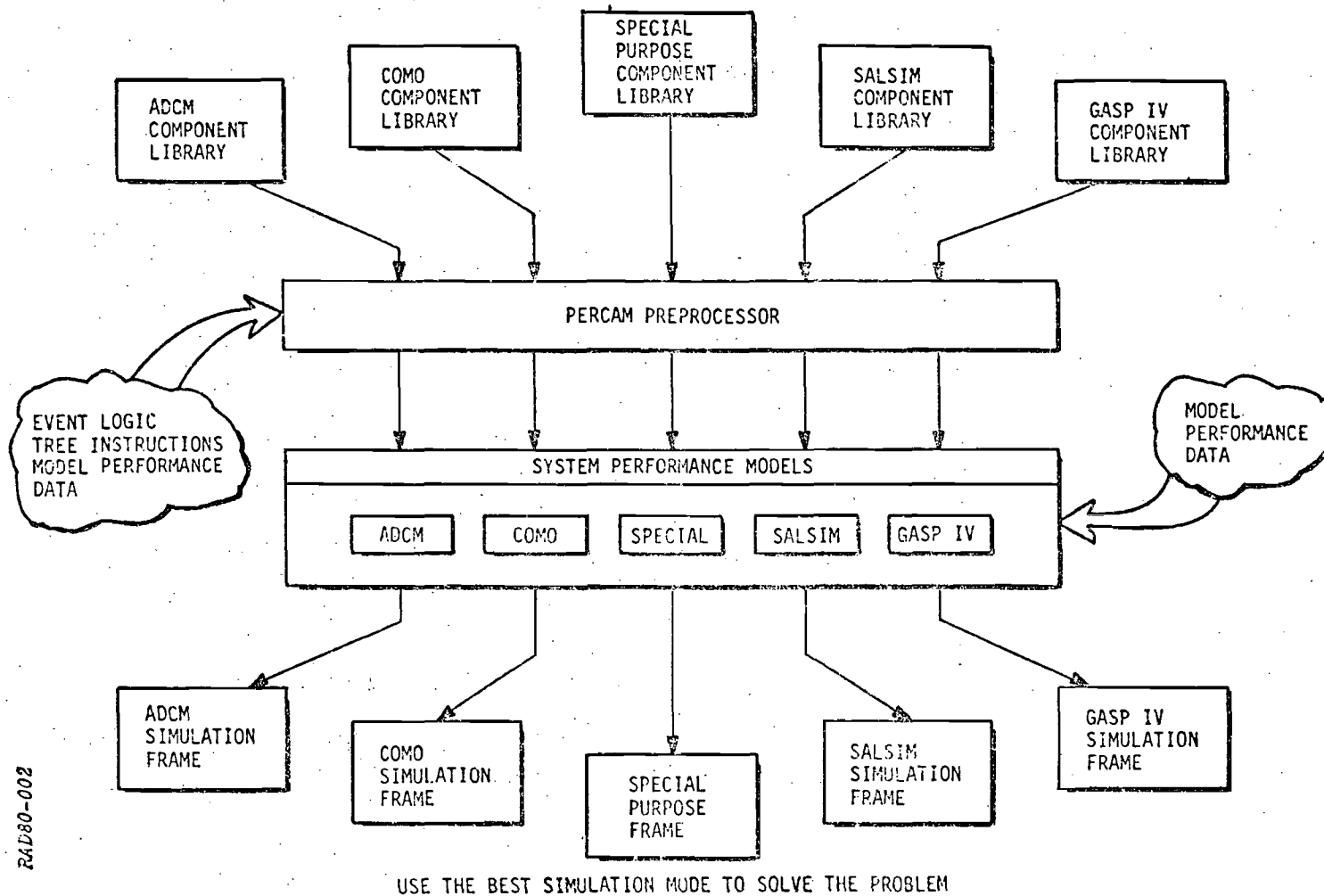


Figure 2-3 PERCAM Multi-Simulator Philosophy

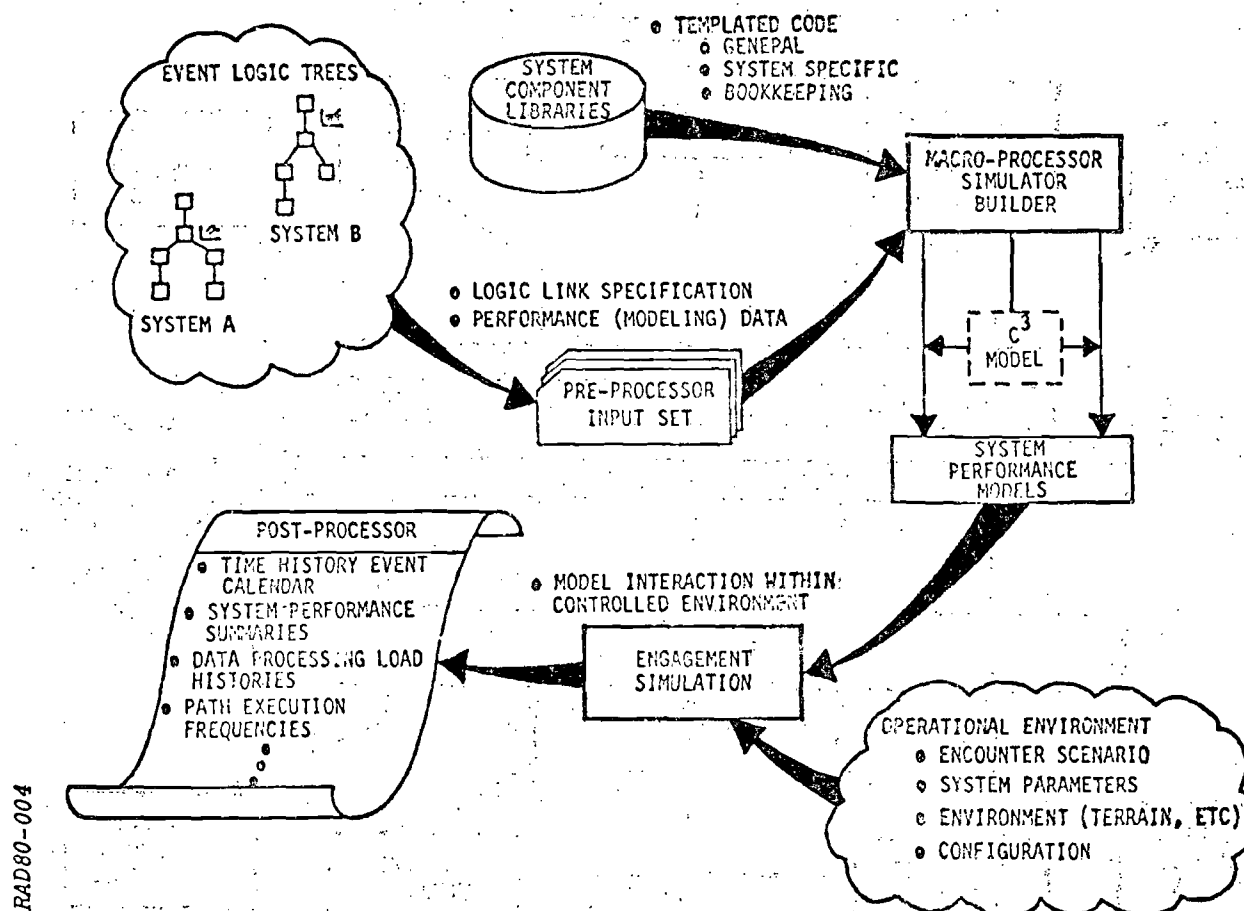


Figure 2-4 The PERCAM Analysis Process

The user will typically execute a series of runs to examine the effects of varying fixed system parameters (feasibility and sensitivity analysis), or perform sets of Monte Carlo runs to determine overall system behavior under non-deterministic conditions. The Post-Processor reduces and displays run outputs for manual analysis by the user. The building-block approach to simulator definition and construction also permits rapid evaluation of alternate system structures and alternate operating rules.

2.3.1 Event Logic Trees

Event Logic Trees (ELTs) are structured graphical representations of the sequence of actions performed by a system in its operating environment. The ELT consists of a series of linked functional blocks that completely describe the operational paths the system may take to reach any number of precisely defined termination points. Branching within the tree is controlled by various decision nodes. An example ELT is shown in Figure 2-5.

Each function is described to the level of detail necessary to represent system operation consistent with the objectives of the study. For example, during the initial phase of a system sensitivity analysis, many areas of the system may be only partially defined. The functional blocks representing those areas might, therefore, be defined with less complexity than other subsystems/areas for which more detailed operational characteristics exist. As the study progresses, more precise operational logic can easily be added within the structure of the ELT.

Associated with each functional element are performance models that define the characteristics of that segment of the tree. It is the performance model that predominately controls the fidelity of an ELT representation. For example, almost all weapon system models require the determination of probability of kill once the defensive ordnance has intercepted the target. The user analyst can control and modify the fidelity of his model to vary the determination of P_k from a simple constant to a complex set of curves that encompass a large number of variables. Once the basic operational ELT structure has been baselined, it is possible to significantly increase (or decrease when desired) the overall modeling fidelity without time-consuming modifications to the logic flow.

Since an Event Logic Tree is a visual representation of a sequence of actions, engineers and systems analysts can easily communicate their understanding of the sensitivity and impact of specific parameters on system performance. In part, a preliminary assessment of the relative sensitivity of each function with respect to the overall performance figures of merit can often be made from visual inspection. First order assessments of this type are very useful as precursor analyses; however, manual analyses are usually too restricted and time-consuming to be used for a full-scale study of a number of different options. An automated simulation capability that builds a computer simulation directly from ELT models has, therefore, been developed to decrease the analysis timeline and provide an effective means for conducting studies that consider a large number of variables.

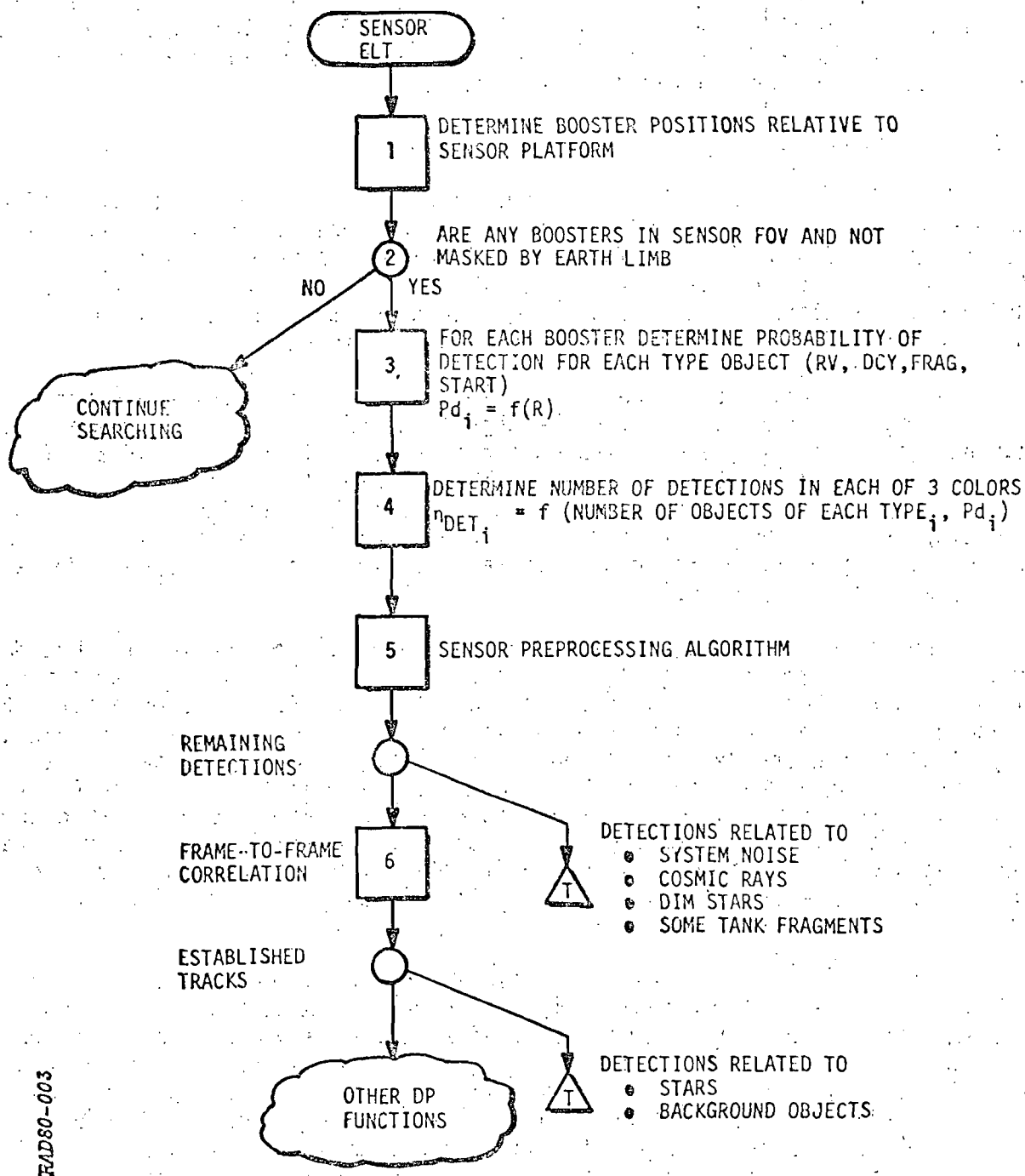


Figure 2-5 ELT Example

2.3.2 PERCAM Software

PERCAM currently consists of four software elements: a Library Builder (LIBLD), a Component Library, a macro-processor simulator builder known as the Preprocessor (PREPRO), and a Post-Processor. Each of these elements is discussed below.

2.3.2.1 Library Builder (LIBLD)

The LIBLD module builds a Component Library of macro templates which is used as one of the two input sets to the Preprocessor. LIBLD also produces printout to describe the Component Library to the user.

A Component Library is created when LIBLD processes a file containing LIBLD Input card image input data. This LIBLD Input describes components which are to be included in the Component Library. Components can (and generally will) contain delimited substitution symbols. These are symbols for which particular text or integers can be substituted when the component is particularized by the Preprocessor. Examples of the three types are component name lines, text lines containing substitution symbols and text lines which contain no substitution symbols. The LIBLD Input corresponding to a single component is composed of one component name line followed by the text lines of the component. The text line card images which follow the name line of the component may be intermixed between text lines which do and text lines which do not contain substitution symbols.

2.3.2.2 Component Library

The heart of the PERCAM system is the PERCAM component library, which provides the templates from which the FORTRAN simulator programs are built. This FORTRAN-based library is flexible, allowing the user of PERCAM to change the components and add new components as dictated by his needs for simulation. The components in the PERCAM library are macro-templates. These templates are FORTRAN statements with macro-processor control characters embedded in them. As the macro-processor itself is very simple, only a knowledge of FORTRAN is required to make extensions and changes to the components. Once the source code of the components to be put into the macro-library is assembled, it is then pre-processed by the component Library Builder into a compact form for efficient access of the modules and quick generation of the simulation program.

The components are intentionally small and generalized to maximize their utility for a wide variety of problems. This decreases the operational efficiency of a simulator, but substantially reduces its construction time, yielding a net time saving to produce output. In practice, the number of basic components for a general purpose library is small. Within a specific technology area, users accumulate more specialized components and problem-particular components over time. Examples of typical components are given in the following subparagraphs.

ADCM - Attacker Defender Control Model. The attacker/defender model component contains the simulation executive, initialization software, and

some of the general support software. Preprocessor inputs for the ADCM component can be divided into several segments:

- Dynamic memory file definition
- ELT declaration
- System performance input specification
- Threat definition
- Program specification.

DC - Decision Component. The Decision Component transfers control to either of two user-specified elements depending on whether the independent variable (P) is less than or greater than a random number, RN, which is in the range 0. to 1.

DPLD - Data Processing Load Specification Component. The DPLD component records data processing throughput, memory, and data rate requirements on an output file for use in post-processing. The requirements for up to ten states or activities can be specified in one DPLD component.

EC - Evaluation Component. The Evaluation Component evaluates a user-specified dependent variable. Evaluation may be carried out in two different ways. In the first, the dependent variable, D, is evaluated as a function of up to four independent variables, VI1,...,VI4. An interpolation is performed using tables supplied by the user. The second evaluation format assigns a value, EXPR, to the user-specified dependent variable. EXPR may be a real constant or a mathematical expression. Both formats may be used simultaneously or independently. The Evaluation Component provides the capability of evaluating a dependent variable as a function of any or several of the program-supplied independent variables. Dependent variables, once evaluated, may be used in other components, e.g., a variable may be input to a Decision Component to determine which of two branches of the ELT to follow.

ELT - ELT Definition Component. The ELT Component must be used to initiate specification of each ELT to be defined in the model. Preprocessor inputs allow the user to define data structures which are local to this ELT. The global data structures defined in the ADCM component are automatically made available. Preprocessor inputs optionally retrieve the pointer to the active system (attacker or defender) from the event notice and store it in a local variable for use in the ELT.

END - End Component. The END component is used to indicate the end of the initialization processing (INITIAL) after the ADCM component and to indicate the end of other subroutines constructed utilizing the component library.

ENDELT - ELT End Component. The ENDELT component is used to indicate the end of a user-defined ELT. Processing on event notice is terminated and the notice is purged from the system.

EVNT - Event Logging Component. The EVNT component is included in the ELT at points where data is to be recorded so that a time history of significant events can be analyzed with the Post-processor. This component causes

data to be written on the Event Tape (TAPE7). Information includes the replication number, game time, event number, event type, active combat unit, and event-related data defined by the user.

MATLB - Math Library Component. This component contains a few commonly used subroutines supporting vector and matrix operation. Included are matrix multiplication, vector cross product, vector product, vector magnitude, vector subtraction, matrix transpose. These routines can be used directly in the FTC/INLINE code. There are no Preprocessor inputs (other than component name) for this component.

TC - Time Component. In addition to allowing one ELT to schedule another (or itself at some future time), provisions have been made with the TC component to allow time delays to occur within an ELT. Processing commences with a designated component within the ELT after the delay has occurred. Specified data (local variables) can be saved for use after the delay has occurred. Any data local to the ELT (i.e., counters) and not designated to be saved are subject to change during the delay period.

FTC/INLINE - FORTRAN Components. The user has two means to augment the component library with FORTRAN code within a model; the FTC component and a pseudo component INLINE.

The FORTRAN Component (FTC) enables the user to construct a specialized code with up to twenty FORTRAN statements -- each of which is not to exceed 20 lines -- ten of which can have statement numbers. These statements can be executable or format statements.

The pseudo component INLINE allows FORTRAN code (in FORTRAN format) to be input in the Preprocessor input stream with the other components. The code input under INLINE is reproduced in the generated code at the point in the ELT model where it was input.

The FTC and INLINE components are provided to allow the user to address needs not anticipated when the other types of components were conceived.

2.3.2.3 PERCAM Preprocessor

The PERCAM Preprocessor assembles the ELT models using components from a Component Library and integrates them with an Engagement Model simulation executive (also contained in the Component Library).

The Preprocessor program uses the Preprocessor input as a guide for generating FORTRAN code which functionally models the attacker and defender. Most of the performance data utilized by PERCAM is input through the Preprocessor.

With the exception of the \$componentname input, which acts as an identifier and causes a particular component type to be evoked from the Component Library, all other inputs cause substitution of a character or character string into the component code replacing the Preprocessor input symbol. If any one of the Preprocessor input symbols in a line of component data is not

specified in the user input to that component, the entire line of code is omitted from the generated code.

2.3.2.4 PERCAM Post-Processor

The PERCAM Post-processor was developed to provide high visibility information on the results and logic processes of PERCAM simulations. This capability has a variety of uses for the analyst. During the test case checkout and preliminary analysis, the event list (which shows the logic followed and decision parameters for each attacker and defender) and the scenario summary (which displays the simulator-received defender and target locations and attacker paths) may be particularly useful. In the comparative analysis phase, the time/range summary (which gives minimum, maximum, and average times for specified defender events) the weapon release summary, and the decision component path execution frequency display (which gives branching statistics) all provide useful, pre-processed data. For results presentation purposes, in addition to the already mentioned displays, there are also the kill probability bar chart, the cost data summary, and a CALCOMP scenario plot.

Since the Post-processor was developed for use with PERCAM, the user input format was constructed to appear as similar as possible to the PERCAM simulator/construction input. Displays are modular and independent of each other and may be requested in any order. Each display request is begun with a keyword, preceded by a \$, as in PERCAM component specification, and other data follows in a free-field format.

The data flow from the user and simulator through the Post-processor is summarized in Figure 2-6. Most Post-processor displays are derived from an event list which is output from the PERCAM simulation. Each component in the component library contains a section of code which writes current data about the component, when executed, on a file (TAPE8). This code is nominally inactive, but may be activated by the inclusion of a control variable in the component specification. Each record written from an executing component is called an 'event'. Also saved are the simulator input data and attacker location time histories.

The modular construction of the Post-processor allows easy addition of new displays. The addition of a new display would nominally involve only the addition of the new keyword to the 'allowable keyword' list, a call to the new subroutine in PROGRAM MAIN, and the inclusion of the new subroutine, or subroutines, in the program library.

2.3.2.5 Future Enhancements

A color graphics display capability was developed on a recent PERCAM project for use in conjunction with the Post-processor. This package is installed at the Army BMDATC Advanced Research Center (ARC) in Huntsville, Alabama, and uses the same ARC Graphics System software that supports REVS. At present, this capability is not formally integrated into the PERCAM system because it is tailored to the ARC facilities. It could be provided for an STE environment since a facility suitable for REVS color graphics would meet PERCAM requirements.

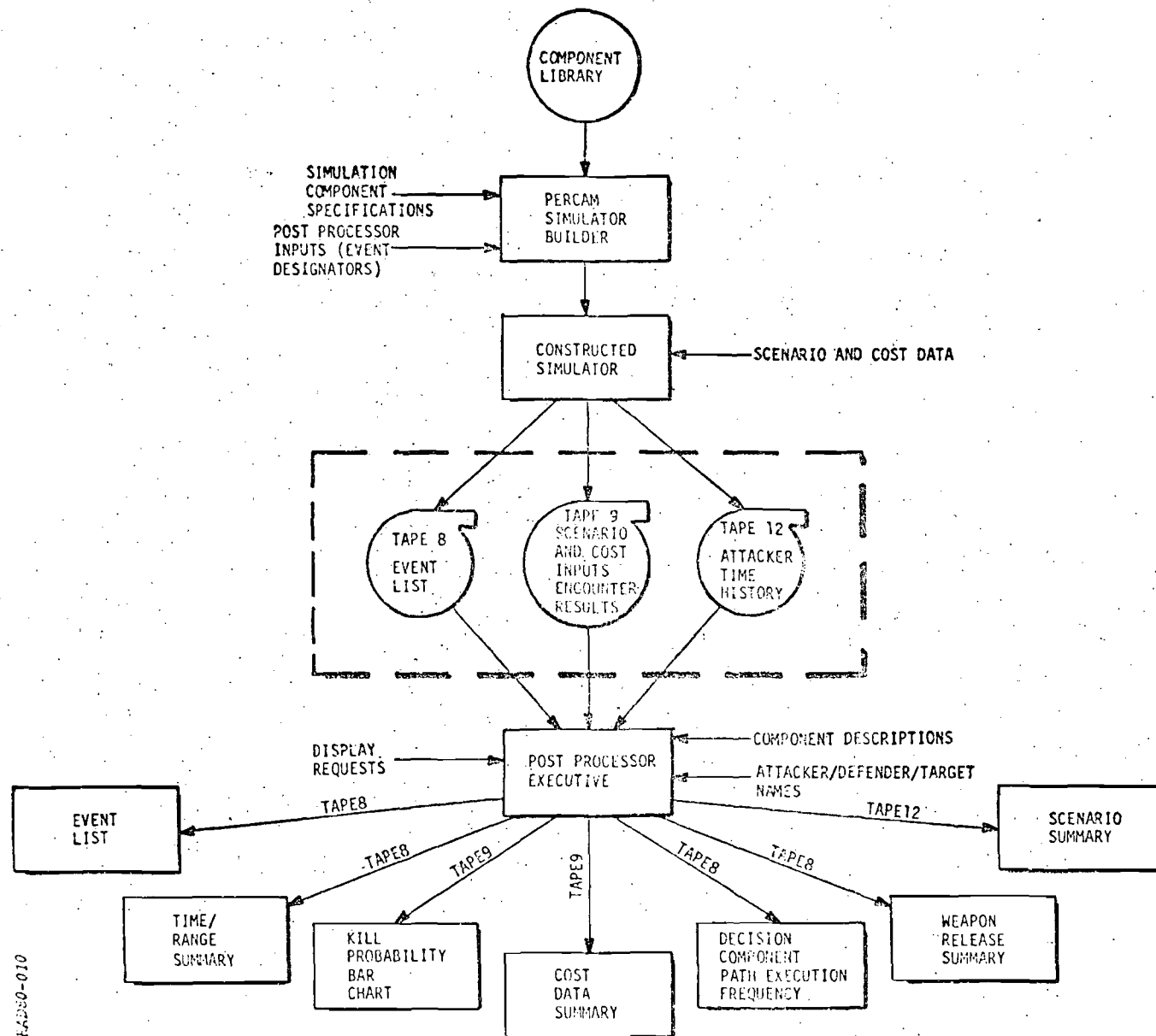


Figure 2-6 User and Simulator Data Input Summary

2.3.3 Current Installations

The PERCAM software was originally implemented on the CDC 7600 and has been used on a variety of CDC 6600 and Cyber series systems. It has also been made operational on both DEC VAX 11/780 and UNIVAC 1100 series computers. It is easily transportable to any computer meeting the software environment requirements of the next paragraph.

2.3.4 PERCAM Software Environment Requirements

While PERCAM is often used in conjunction with specialized simulation languages (e.g., COMO, SALSIM) we will not require support of those languages for a basic capability. All that is needed to achieve a PERCAM environment is a suitable extended FORTRAN IV (FORTRAN 66) compiler and specific system utilities as itemized in the following subparagraphs. Future color graphics requirements can be met within the REVS graphics requirements.

2.3.4.1 Required FORTRAN Language Extensions

- Additional characters for the FORTRAN Character Set are required as special characters to the Preprocessor. These are used to inform the Preprocessor of parameter substitution names and of literal character strings.
 - Single quote (') in addition to the FORTRAN double quote (").
 - The Less than character (<).
 - The Greater than character (>).
- Namelist I/O. Namelist I/O is a non-standard FORTRAN feature. This non-standard extension may not be the same on different host systems. Namelist I/O is required for the convenience of changing identifier names simply by specifying the name and the value to be replaced. This is used very heavily in PERCAM for revising initial conditions, probabilities, and other types of operations.

2.3.4.2 Required System Utilities

PERCAM requires the following system utilities, in addition to the FORTRAN compiling system:

- Sort/Merge. A Sort/Merge package for the host system must be callable by FORTRAN. This capability is required for the sorting done in building the table data summaries of the Post-processor.
- CALCOMP Plotting. The CALCOMP basic plot routines or equivalent are required for PERCAM outputs.

- Multiple Job Sequence. The job control language of the host system must provide for multiple job step execution. A typical PERCAM run executes in the following order in one job stream.
 - Library Processor (LIBLD)
 - Preprocessor (PREPRO)
 - FORTRAN Compiler
 - Loader
 - FORTRAN Simulation
 - Execution
 - Post-Processor.

Thus, multi-phase processing must be provided by the host system. The user must be able to execute the sequence in full or in increments during any one run.

- Check-Point Restart. For very large scenarios involving lengthy run times it is desirable to save data at intermediate check-points sufficient for the run to be restarted at the last check-point prior to host system failure. This ensures that lengthy reruns would be unnecessary. While this feature is useful for theater-wide air defense modeling, we do not envision it as necessary for the near-term STE unless the host slowdown factor relative to current installations is large.

2.4 AUTOIDEF*

AUTOIDEF is an interactive graphics tool which provides a means of designing in the ICAM Definition Method (IDEF). AUTOIDEF is being developed by Boeing Computer Services Company (BCS) under contract to SofTech, Inc.

AUTOIDEF is designed to provide users with a computer-supported means of graphically representing the activities, data constraints, and dependencies in a manufacturing architecture. Phase 1 of the three-phase AUTOIDEF project provides for the design of IDEF diagram construction and limited editing according to a user's set of specifications, and includes the capability of viewing or plotting any IDEF diagram following its construction. AUTOIDEF users will be able to save diagrams and entire data bases, and recall them at a later time. The Build 1 version of AUTOIDEF also enables the user to perform the administrative functions necessary to establish new projects and to specify new authors and users. Build 2 will result in a complete diagram input, editing, and display capability, as well as model creation and display capabilities and project management support. Build 3 will yield a set of model analysis tools, information extraction tools, and alternate format presentation tools.

*Note: Information and illustrations in these sections have been excerpted and adapted from ICAM project documentation provided in part by Boeing Computer Services, Tukwilla, Washington.

It should further be understood that IDEF is an evolving methodology. It is expected that some revision will be made, and that the corresponding revisions to the tools will be made prior to Build 3.

Figure 2-7 illustrates a typical IDEF diagram. The allowable diagram structures and relationships dictate a graphics mode of diagram definition. Since multiple arcs may originate or terminate at box boundaries, and arcs may fork or join in complex ways, specification of a diagram requires detailed description of graphic coordinates. Entry of descriptions via input cards would be either tedious and impractical, or would constrain the allowable structure to a limited set.

2.4.1 AUTOIDEF Software

The structure of AUTOIDEF Build 1 software consists of eleven distinct functional components as shown in Figure 2-8. The implemented software has been modularized to enforce these functional boundaries. The overall architecture is expected to remain constant in Build 2. The following subparagraphs provide functional descriptions of each component.

2.4.1.1 AUTOIDEF Executive

The AUTOIDEF Executive component controls the entire system. It is responsible for scheduling and invoking the Administrative Functions and Diagram Executive/User Interface components. Depending on the user's needs the Executive will invoke one of these subordinate components and, for the Diagram Executive, will also initialize the use of a graphics terminal. When interfacing with the AUTOIDEF Executive, the user can perform administrative functions, such as specifying authors and additional users, using any keyboard terminal.

The AUTOIDEF Executive determines the options available to a given user. It monitors the use of IDEF diagrams to ensure that only those diagrams for which the user is an author or specified user are accessible.

2.4.1.2 Administrative Functions

An AUTOIDEF user has a choice of several functions through the Administrative Functions component, consistent with his access authorization. For Build 1, these administrative functions are: create a new author, specify additional users, and delete a diagram. According to the IDEF conceptual schema, entities are either created or deleted from the data base, depending upon the function.

2.4.1.3 Diagram Executive/User Interface

A user enters the Diagram Executive/User Interface component to display, plot, create, or edit an IDEF diagram. The user is presented with options via a menu selection interface and interacts with AUTOIDEF by making a choice. The Diagram Executive/User Interface component interfaces with a graphics terminal once initialization is performed by the AUTOIDEF Executive. The component then interfaces with the Diagram Transfer and Diagram Manipulation components to perform the activity selected by the user. The Build 1 AUTOIDEF

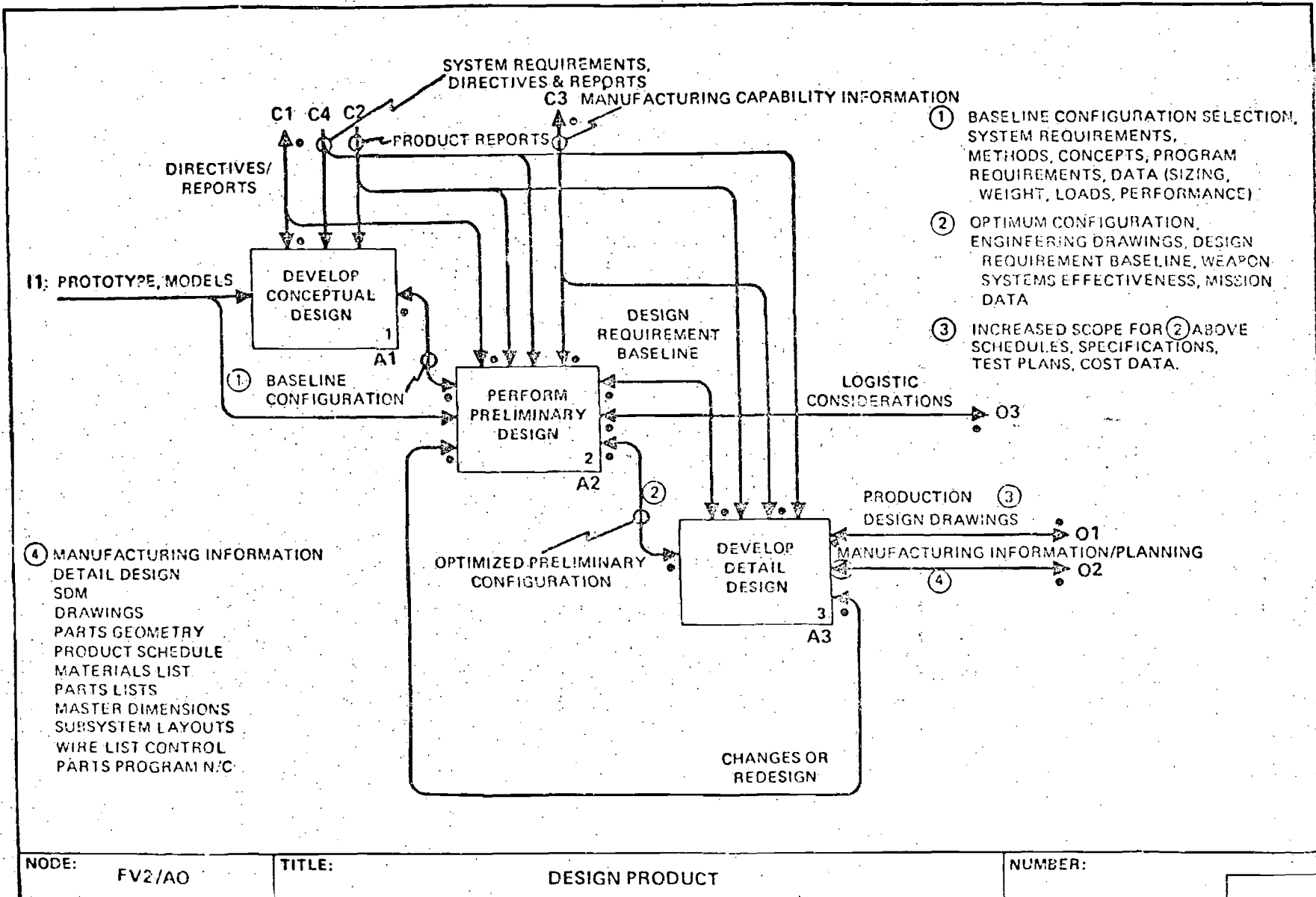


Figure 2-7 Example IDEF Diagram

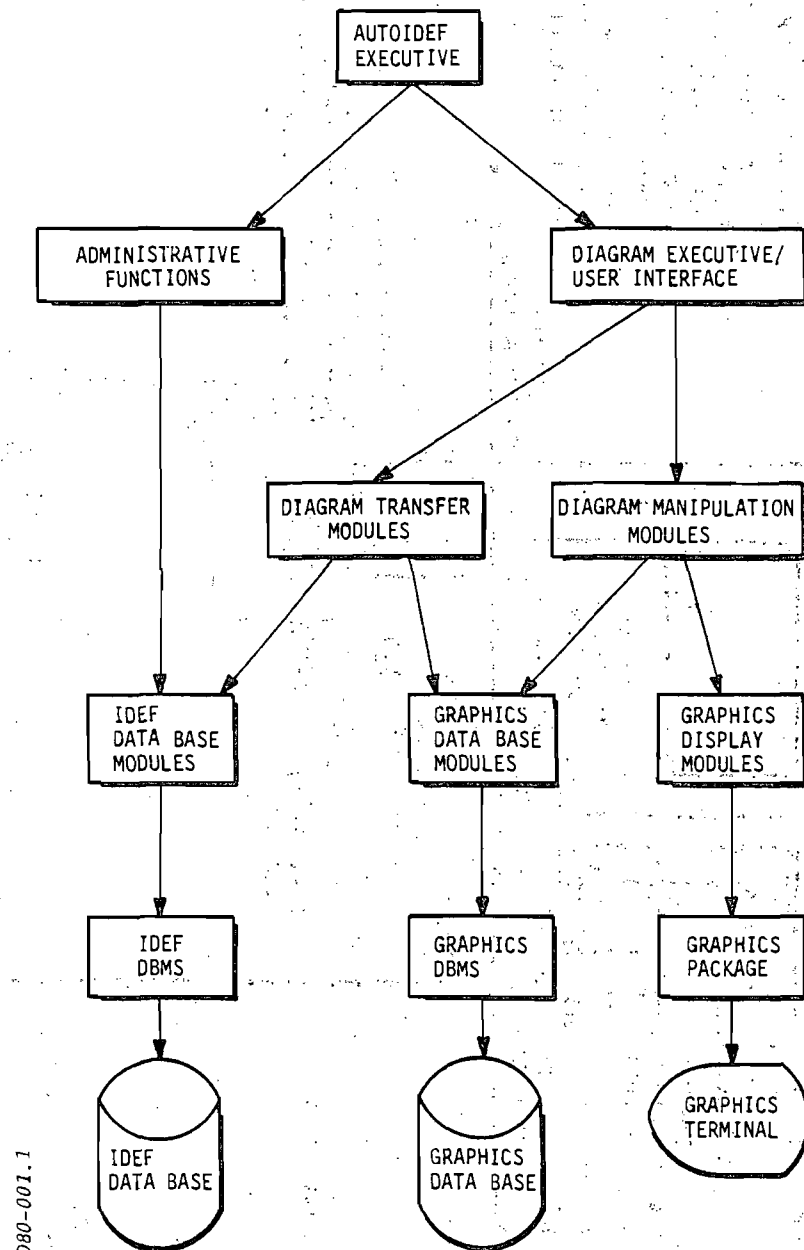


Figure 2-8 AUTOIDEF System Overview

Diagram Executive/User Interface component is based upon the menu structure of the AIDS*Staging system.

2.4.1.4 Diagram Manipulation

The Diagram Manipulation component performs the graphical manipulation of an IDEF diagram. The manipulation routines are of three types: diagram construction, diagram display, and plot diagram. Diagram construction allows a user to manipulate either an existing or a new diagram by performing such functions as add N boxes, add an arrow segment, add an ICAM code, or add an arrow segment label. Diagram display and plot diagram generate either a screen display or a hardcopy plot of an IDEF diagram. The Diagram Manipulation component is invoked by the Diagram Executive/User Interface component and in turn must interface with both the Graphics Display and Graphics Data Base components.

2.4.1.5 Diagram Transfers

The AUTOIDEF Diagram Transfer component controls the transfer of information between the graphics data base and the IDEF data base. The transfer process is based on the IDEF conceptual schema developed during preliminary design of AUTOIDEF. IDEF diagram information is passed from the IDEF data base to the graphics data base when a user desires to perform diagram manipulation. For a new diagram, only author and project information is transferred; for editing an existing diagram, all pertinent information concerning the diagram's related entities is transferred to the graphics data base. Once a diagram has been displayed, plotted, created, or edited in the graphics of AUTOIDEF, the information is returned to the IDEF data base by the Diagram Transfer component.

The Diagram Transfer interfaces with both the IDEF and Graphics Data Base components to accomplish the transfer process. The Diagram Manipulation component is responsible for invoking the Diagram Transfer component moving of the diagram information.

2.4.1.6 IDEF Data Base

The IDEF Data Base component has two levels of capabilities. The first-level data base routines make use of the IDEF conceptual schema designed during the preliminary investigation. The conceptual schema is then translated into an implementation schema by both Level 1 and Level 2 data base routines. The IDEF data base implementation schema is based upon a CODASYL structure used by ADBMS**, the current IDEF DBMS. The logical data base structure maintained by ADBMS is compatible with the entities, attributes,

*AIDS is an interactive graphics-based system developed at Battelle-Columbus to help in the preliminary design of aircraft. The User Interface, Graphics, and Data Base routines are being used in support of AUTOIDEF.

**ADBMS means "A Data Base Management System". It was developed by the University of Michigan ISDOS project, based on DBTG71 CODASYL model. The current IDEF DBMS uses ADBMS Version 3. The REVS DBMS was adapted from ADBMS Version 2.

and relations that support the IDEF methodology. The IDEF data base implementation schema is used by ADBMS to perform the physical processing necessary to automate the IDEF methodology. Both Levels 1 and 2 routines are designed with generalized interfaces. Level 1 interfaces with both the Diagram Transfer and the Administrative function components through an interface based on the conceptual schema. Level 2 is then used to format and control ADBMS routines in order to accomplish conceptual requests based upon the current implementation schema.

2.4.1.7 IDEF Data Base Management System

ADBMS, the current IDEF DBMS, presents a generalized data base structure based on a CODASYL schema. The CODASYL schema is translated to meet the needs of the IDEF methodology resulting in the IDEF data base implementation schema. The implementation schema is then used by the IDEF data base modules to respond to user requests from other AUTOIDEF components. To the DBMS, the implementation schema represents the logical structure of the data. This logical structure is used to perform the actual physical access of data. The physical structure of the data is unimportant to any of the AUTOIDEF components other than the IDEF DBMS component.

2.4.1.8 Graphics Data Base

The Graphics Data Base component is similar to the IDEF Data Base component. It makes use of both a conceptual and an implementation schema by separating the module into two levels. The conceptual schema is an extraction of the IDEF conceptual schema and represents only the entities related to a given diagram. The implementation schema is based upon the logical structure maintained by the AIDS-Staging system rather than ADBMS in the IDEF data base implementation schema. Again, a transformation process is responsible for interpreting the conceptual schema to the implementation schema. The implementation schema is then used by the Graphics DBMS to perform the actual physical accessing of the data. The Graphics Data Base component interfaces directly with the graphics DBMS through the implementation schema and also interfaces with the Diagram Transfer component and Diagram Manipulation component by using the conceptual schema.

2.4.1.9 Graphics DBMS

The AIDS-Staging system is the current graphics DBMS in AUTOIDEF. AIDS uses a leveled logical data base structure that is tailored to the IDEF methodology to become the implementation schema. The Graphics DBMS is responsible for manipulating this implementation schema to physically access the data relevant to an IDEF diagram. Only the Graphics DBMS needs to have knowledge of the algorithms and structures necessary to perform the actual physical accessing. The Graphics DBMS interfaces only with the Graphics Data Base component through the Graphics implementation schema.

2.4.1.10 Graphics Display

The Graphics Display component has three major functional capabilities: generalized entity descriptions, verification and monitoring of the AUTOIDEF screen display, and interfacing with the Diagram Manipulation component and the Graphics Package. The entity description function is used by AUTOIDEF to make the display of an IDEF diagram more efficient. Several data items are maintained to describe the dimensions of boxes, the IDEF form, and locations of identification information fields. The Graphics Display component also verifies and monitors the current contents of the graphics screen so that the construction of IDEF diagrams is valid. The screen verification process ensures that arrows do not cross boxes, boxes do not overlap other boxes, and arrow segments are gapped when they cross labels. The Graphics Display component interfaces with both the Diagram Manipulation and the Graphics Package components. The interface to the Diagram Manipulation component is structured around the following subfunctions: open a display segment, display text or lines, and close the display segment. The low-level interface performs the same functions but also includes a mapping to core graphics.

2.4.1.11 Graphics Package

The major purpose of the graphics package is to maintain a graphics display file that represents an IDEF diagram. The graphics package, AIDS-Inter-tek, is responsible for creating, deleting and maintaining the display file. The AUTOIDEF graphics package interface is mapped to meet core graphics standards, which can be used on many graphics packages. The graphics display file is used both in displaying an IDEF diagram and in plotting a diagram. The only difference in the functions is the output device the display file is directed to: graphics terminal or hardcopy plotter.

2.4.2 Current Installations

The AUTOIDEF system is currently installed on the CDC CYBERNET system under the NOS operating system. Users throughout the country can access AUTOIDEF through local CYBERNET concentrators.

Eventual rehosting of the software is probable when the ICAM project progresses to the point of fielding a completely integrated set of tools. With this in mind, AUTOIDEF has been designed for the maximum degree of portability possible, consistent with its detailed capability requirements and the requirement to initially host it on CDC CYBERNET equipment.

2.4.3 AUTOIDEF Software Environment Requirements

The subparagraphs below summarize software environment requirements as identified from currently available AUTOIDEF documentation. Future builds of AUTOIDEF are not believed to involve additional requirements.

2.4.3.1 FORTRAN Language Extensions

The AUTOIDEF software and its externally-furnished supporting packages are stated to be in conformance with the ANSI FORTRAN 77 standard except for certain identified deviations; most of which seem related to the CDC environment and are well confined to specific submodules. The only major deviation involves use of MASK, SHIFT, ENCODE, and DECODE functions. These must be provided in the STE environment to avoid rework of AUTOIDEF.

In addition to use of the functions above, AUTOIDEF may deviate from the older ANSI FORTRAN 66 standard in the following:

- Operand conflicts in arithmetic statements.
- Use of literals instead of Hollerith data.
- Use of ENTRY statements.

Although the DBMS from the University of Michigan is claimed to be in ANSI FORTRAN, it supports random access I/O which is a deviation from the FORTRAN 66 standard (and is specified in the FORTRAN 77 standard in a manner that deviates from most known implementations). A candidate STE host must support FORTRAN random access I/O. The implementation on each candidate system must be evaluated to determine compatibility as there are often undocumented variations between machines.

2.4.3.2 Assembly Languages

The AUTOIDEF documentation states that one routine in the AIDS executive is written in COMPASS (CDC Assembly language). This routine will have to be rewritten in either host assembly language or HOL for non-CDC machines.

2.4.3.3 Character Set

Due to limitations of the CDC NOS operating system, AUTOIDEF users are currently limited to the ASCII 64 character set (upper case). However, the original AUTOIDEF requirements state that it is desirable that the full 96 character ASCII character set be available. The STE host should provide the 96 character set, but the 64 character set would be acceptable if other considerations favor selection of a particular machine.

2.4.3.4 Memory Management

As with REVS, in the CDC environment AUTOIDEF uses the CDC segmented loader to minimize the use of main memory during execution. AUTOIDEF is designed to allow segments no larger than 25676 60-bit words each (60K octal). These segments are overlaid as necessary during program execution. Any overlay generation mechanism used by the STE host must be transparent to the application program. A virtual memory system is an acceptable alternative to segmentation and overlays.

2.4.3.5 File Access

The original AUTOIDEF requirements specified that read/write access to the data base be provided for multiple users simultaneously. The CDC NOS operating system does not support access by more than one user at a time. To work around this constraint, the designers chose to segment the data base among multiple physical files to provide data base access for multiple users. Each user may access only one file for read/write operations. No two users can access the same file concurrently for write operations. To support this mechanism, the STE host must allow AUTOIDEF to dynamically attach, open, close, and release files during program execution, in response to user command.

2.4.3.6 Alternate Graphics Packages

BCS states that other graphics software packages may be substituted for the AIDS-Intertek FORTRAN package currently used if they provide the functions "provided in the Core graphics mapping" (defined in an appendix not included with our documentation). We presume this implies compatibility with the Core System, a graphics standard developed by the ACM/SIGGRAPH Graphics Standards Planning Committee (see ACM Computing Surveys, Volume 10, No. 4, December 1978 for a discussion of the Core System).

2.5 GENERAL SOFTWARE ENVIRONMENT REQUIREMENTS

This paragraph addresses software environment capabilities needed for the STE independent of the specific tools. These features are characteristic of any effective development environment.

2.5.1 Source Library Control Requirements

Because of the size and complexity of the REVS and AUTOIDEF systems, some means of source library control must be provided. An operating source library system must:

- Logically group source modules.
- Provide a method of source updating to include an audit trail.
- Provide a means for maintaining different software versions for different site variations.

REVS currently uses a source library system on the CDC. Unfortunately, the CDC UPDATE program is not machine-independent, it is applicable only for CDC sites. The CDC UPDATE provides a proper audit trail so that previous additions can be identified, and updates are accomplished in a non-destructive mode. Finally, the CDC UPDATE system logically groups the source modules of the application system. Two approaches for source library control are possible. Either the host computer's library system can be used or a "portable" system (i.e., dependent only on character set standards, such as ASCII) can be used.

2.5.2 Data Entry Requirements

The STE must effectively support data file creation, editing, update, and deletion operations in both batch and on-line modes. This support must be independent of the hosted tools. The separate files must be mergeable and attachable to execution jobs.

2.5.3 On-Line Response Time Support

The STE host operating system must be capable of supporting a 1 to 4 second reaction time to local on-line jobs. This means direct or indirect acknowledgement of a request from an on-line terminal, and not necessarily completion of the requested action. (This reaction time is generally considered adequate for interactive systems [8]). This response time must be supported in a multi-user environment (up to six, but nominally two or three STE users).

2.5.4 Communications Support

The STE systems software must be capable of servicing up to six communication lines from either local or remote terminals at transmission rates up to at least 4800 baud (9600 baud preferred).

3.0 HARDWARE RESOURCE REQUIREMENTS

The paragraphs of this section discuss the hardware resources necessary to support REVS, PERCAM, and AUTOIDEF. Of the three, REVS and AUTOIDEF appear to set the pacing requirements for an STE facility. The needs of PERCAM are easily met within this framework. In the final paragraph of this section, we discuss general issues in STE hardware selection.

3.1 REVS RESOURCE REQUIREMENTS

REVS was originally developed for the TI-ASC and rehosted on the CDC 7600. Subsequent improvements in performance have enabled practical installations on slower CDC Cyber 170 series machines. Currently, REVS is being rehosted on a yet slower machine, the DEC VAX 11/780. This paragraph describes REVS resource requirements for the STE environment based on past and current experience.

3.1.1 CPU Speed

The basic CPU speed of a candidate STE configuration is only one aspect of potential REVS performance, to be considered concurrently with primary memory size, I/O transfer efficiency, total cost, and user demand. Therefore, we will set no specific CPU speed requirement, but will discuss REVS performance in terms of "slowdown" relative to current installations.

The fastest version of REVS is on the CDC 7600 at the BMDATC ARC. Most REVS runs use less than 30 CPU seconds on this machine, with 8 to 20 seconds a typical value for experimental exercises of the type probably prevalent at the STE during the R&D period.

To explore the maximum relative slowdown value, however, we should consider the longest runs that may be required at the STE. One such run is the creation of the REVS load module itself, during the installation phase (see Table 4.1 in the next section). This is the second longest single run in the installation phase, and uses about 120 CPU seconds on the CDC 7600.

The longest run in the installation phase is a test case that uses 192 CPU seconds. This case involves the creation and listing of a complete data base for an air defense system. It is a realistic representation of a "large scale" software requirements engineering problem and uses the maximum LCM allowance at the ARC. A larger data base at the ARC would require paging from disk. Few REVS data bases will reach this size. Although not mandatory, any data bases that exceed this size would generally be partitioned into smaller data bases for division of labor and management resources. Of the 192 CPU seconds, 131 were required to create the data base from card images, and 55 seconds were required to list it. The remaining six seconds were for REVS and system setup and termination activity. The run produced nearly 24,000 lines of output, and used over 10,000 card images for input.

The same test case required 1064 seconds to execute on the TRW Time Sharing System (TSS) Cyber 174 configuration. The data base creation (translation) time was roughly 759 CP seconds while the listing time was approximately 305 CP seconds. This is a slowdown of about 5.5:1 relative to the 7600.

An additional order of magnitude slowdown (i.e., 55:1 relative to the 7600) would mean that this run would execute in about three hours. Production of a REVS load module would take about 110 minutes. Execution of the REVS verification test cases would take about four hours and 40 minutes. Most typical REVS runs would be in the seven to eighteen minute range. For future STE candidate evaluation purposes, a 55:1 slowdown would seem to be a maximum outer limit, but only for an STE-dedicated facility. The central issues would be cost and the ability to meet overall daily needs within this limit.

Certainly in a shared facility, such as an existing ARPANET node, a 55:1 ratio would be unacceptable as it would deny the node to other users. Here, ratios in the range of 5:1 to 12:1 would be more practical, subject to availability considerations.

3.1.2 Primary Storage

The "smallest" installation of REVS to date has been on the "B" machine at the Naval Air Development Center (NADC). This configuration is considered to be very limited in performance and does not include interactive graphics. This version required 130000_g CM (central memory) 60-bit words to execute with an additional 144000_g words of ECS (Extended Core Storage -- essentially fast disk). If we were to consider all of this to be primary memory, we can assume that 0.8 Mbytes would afford a very limited REVS configuration that would be totally unacceptable with CPUs significantly slower than CDC Cyber 175's. The configuration on the NADC "C" machine is equivalent to 1.45 Mbytes and is considered a "good" configuration. The best configuration is at the BMDATC ARC. It is equivalent to 1.5 Mbytes, but requires less I/O activity and is more efficient because up to 128 Kwords (~ 1 Mbyte) of user data base is stored in Large Core Memory (LCM) which is considerably faster than ECS. All of these versions utilize code overlays to minimize the amount of REVS software in core. If we "unrolled" the code, for instance to allow multi-user reentrant access, and maintained it in primary memory, about 1.5 Mbytes would be needed for REVS software with additional requirements for user data base space. Making allowance for four concurrent users we could visualize a very fast version of REVS with 4 to 6 Mbytes of primary memory.

However, we believe that approximately 2 Mbytes is a desirable primary storage figure for the STE that would support further REVS performance trade-offs and allow margin for other host system functions as well as future tool growth. As little as 1 Mbyte might be acceptable if I/O transfer was extremely efficient. Accordingly, we will set a cut-off threshold at 1 Mbyte for potential STE candidate hosts.

To illustrate the impact of primary memory limitations on I/O transfers, we will cite the air defense example mentioned in 3.1.1. The data base for this system is about 1.1 Mbytes. On the CDC 7600 at the BMDATC ARC, virtually all of this data base is contained in LCM. On the TRW TSS Cyber configuration, only a fraction of the data is in central memory. For the test case previously cited, I/O activity at the ARC was 7.4 Mbytes, while on TSS it leaped to 1063 Mbytes. Because of the long execution time on TSS, much of this activity was undoubtedly also due to job swapping.

3.1.3 Mass Storage

The STE production environment must allow for REVS software files, object code mass storage, plus adequate allowance for active and archived user data bases. Table 3.1 shows a breakdown of the minimum production phase mass storage for the REVS software and ancillary files (3.35 Mbytes). To this should be added space for up to six active projects and ten archived projects.

REVS project data bases vary considerably in final size, and of course vary in size during development. One of the larger data bases we have developed so far, for the air defense system mentioned previously, is about 1.1 Mbyte in size. While many data bases for operational systems might reach that size, experimental data bases (and those for most systems) are generally more compact. The assumption of a 0.5 Mbyte final data base size is realistic for mass storage sizing.

In practice, it has been found that an active mass storage allocation of about five times the existing data base size is desirable during the early phases of a project with a reduction to three times data base size permissible as the data base nears completion. However, allowance must be made for storage of simulators and post-processors when they are used. We, thus, are recommending that the minimum allocation be maintained at five times data base size throughout the project for STE purposes. This provides adequate room for creation of new data base versions and retention of selected previous data base versions.

Conservatively using a 0.5 Mbyte size and multiplying by 5, the average project allowance should be 2.5 Mbytes. Allowing for sixteen data bases (six active projects, ten inactive) yields a minimum mass storage allowance of 43.35 Mbytes, after adding in the 3.35 Mbytes for REVS software. Additional allowance must be made for user I/O files and host system needs.

During installation of the REVS, 12 Mbytes of mass storage is needed for the software, as shown in Table 3.2. Installation of a modified REVS concurrent with production use of a previous version could require up to 12 Mbytes over the production needs estimated above.

Finally, if the STE is to be used for further development and modification of REVS concurrent with production use and evaluation, extra mass storage allowance must be made for a REVS development phase. At most, this would be twice the installation allowance plus 20 percent, or 29 Mbytes, but could be as low as 15 Mbytes. For estimation purposes, we will use the higher number.

Table 3.1 Production Phase Mass Storage Files
For REVS Software

FILE	CDC 60-BIT WORDS
TRW REVS 7600 SYSTEM	103,102
• PASCAL LIBRARY	
• PASCAL COMPILER	
• NESTER	
• DATA BASE LIBRARY	
REVS 200	277,929
• SEGMENTED CODE OF REVS SYSTEM	
VVDB	2,703
• EMPTY SIMULATION DATA BASE	
VVDBT	156
• SIMULATION DATA BASE TABLE	
ASSMDBRSL NUCLEUS	13,566
• NOMINAL INITIAL RSL DATA BASE	
ASSMDBT	823
• RSL DATA BASE TABLE	
DONNEES	7,852
• RSL TRANSLATOR INITIALIZATION	
RISF	7,080
• CODE PIECES FOR SIMULATOR	
VVDBLDR	32,870
• SIMULATION DATA BASE BUILDER	
TOTAL	446,081*
*(APPROXIMATELY 3.35 MBYTES)	

RAD80-016

Table 3.2 Installation Phase Mass Storage Files
For REVS Software

FILE	CDC 60-BIT WORDS
CWSPC	59,395
• COMPILER WRITING SYSTEM SOURCE	
PASCAL SOURCE 7600	69,437
• PASCAL COMPILER & LIBRARY SOURCE	
REVSCDCSOURCE 2X(308040)	616,080
• REVS SOURCE	
• TWO COPIES REQUIRED FOR CWS UPDATE	
REVSDBCSSOURCE	38,882
• SOURCE OF DATA BASE CONTROL SYSTEM	
REVSTESTCASES	82,992
• REVS RSL TEST CASES	
ASSMDB	2,703
• EMPTY ASSM DATA BASE	
PLIB	46,701
• PASCAL LIBRARY	
REVSDBCSLIB	51,224
• DBCS LIBRARY	
REVSLIB	56,242
• REVS OBJECT CODE LIBRARY	
REVSMAC	3,557
• REVS JOB CONTROL PROGRAM	
VVLIBE	4,240
• TEMP OBJECT LIBRARY	
SUBTOTAL	1,031,453
PRODUCTION FILES FROM TABLE 3.1	446,081
TOTAL	1,477,574*
*(APPROXIMATELY 12 MBYTES)	

RAD80-015

Thus, the STE should provide for 72 Mbytes (43 + 20) of mass storage associated with REVS development and use, with additional allowance for user I/O files and host system needs. The latter must be assessed for the STE as a whole, and in conjunction with evaluation of the STE candidates.

3.1.4 REVS Interactive Color Graphics Requirements

This paragraph discusses the current REVS graphics environment and interfaces at the BMDATC ARC, and outlines color graphics support requirements for the STE.

3.1.4.1 ARC Graphics System Hardware

The ARC Graphics System consists of four interactive terminal sets connected through special-purpose interface hardware and software to the CDC 6400/7600 data processing system located in the ARC facility in Huntsville, Alabama. Each terminal set includes a 19-inch color CRT, keyboard, and trackball positioned cursor. The system is supplemented by a large screen video projector and a black and white hard copy device. Terminal operation is initiated and controlled by time-shared batch jobs executing in the CDC 6400/7600 computer system. User access to the graphics terminals is supported by a package of FORTRAN callable subroutines that generate display output commands and receive keyboard and trackball cursor inputs.

The major hardware components of the ARC Graphics System are:

- Communications processor
- Anagraph display system
- Large screen video projector
- Hard-copy unit.

The current ARC configuration is no longer commercially available. A modern equivalent would be the Ramtek 9400 Graphics Display System. A minimum price option version supporting two terminals would cost between \$50K and \$100K. Such a system could support only local on-site use.

3.1.4.2 Proposed STE REVS Color Graphics Environment Requirements

The STE should permit simultaneous connection, via local or remote RS-232-C full duplex serial data interface, of at least two interactive color graphics terminals at a line rate of 4800 baud or higher. The STE graphics terminals should provide a serial RS-232-C full duplex interface with user-selectable transmit/receive rates up to the maximum rate. Each terminal should provide an EIA-RS-330 closed-circuit television standard interface for attachment of hard copy units/video monitors/video recorders.

The STE graphics terminals should provide a display with 480 x 640 point resolution and should be capable of simultaneously displaying eight distinct colors under application program control. Screen diagonal size should be greater than or equal to 13 inches.

The STE graphics terminals should provide sufficient memory to store a full display of graphics data. Adequacy must be determined according to the display definition requirements of the particular terminal as estimated against REVS graphics display benchmarks (to be determined).

The display should support the standard ASCII 96 upper case/lower case character set at minimum. Graphics software provided by the vendor should be capable of interfacing with FORTRAN or Pascal software residing on the STE host computer. The graphics display should provide a user-steerable input cursor, controlled by trackball, keys, or joystick and should allow for alphanumeric data entry concurrent with graphic display.

Other desirable features to be evaluated in the selection of a graphics display terminal are:

- Cost \leq \$20K configured per requirements.
- Additional user aids (e.g., ruling characters, patterns, forms/data entry aids).
- Support for geometric shapes and special symbols.

The STE graphics support peripherals should include a video hardcopy unit, capable of interfacing with multiple graphics terminals, to output 8-1/2 by 11 black and white hard copies of terminal displays.

3.1.5 Plotter Hardware

REVS currently makes use of CALCOMP continuous roll-feed plotters, with both 12 inch and 30 inch width paper at the BMDATC ARC. REVS produces a plot input tape so that the plotters can be operated off-line.

The large-size plots are used as working drawings, but are generally too large for inclusion in documentation. Plots on 12 inch paper (height) can be used on fold-out pages, if not excessively wide, and on standard 8-1/2 by 11 pages for the majority of instances. Future REVS development will focus on making 8-1/2 by 11 inch plots more legible and useful in publications.

While the continuous-feed plotters and optional 30 inch width are desirable features, they need not be mandatory for the STE. However, an STE plotter must produce at least 8-1/2 by 11 inch plots and accept tape input in CALCOMP compatible format.

3.1.6 Card Reader/Alphanumeric CRT Terminals

Inputs to REVS vary from a few card images up to 2000 card images under usual working conditions. The larger input sets are used in the earlier stages of data base construction.

With the trend to "cardless" data entry environments, the card reader is becoming more of an optional extra than a necessity. For the STE, this is also true. REVS inputs can be handled directly from cards, indirectly

via creation of input files using an alphanumeric CRT terminal, or directly via the interactive graphics terminal. The STE should provide the capability to construct and edit input files without having to invoke REVS.

The only real requirement on a card reader or CRT terminal, for input, is that it be able to accept standard 80-column card images and transfer them in a form acceptable to the selected STE host computer. In the case of CRT terminals, the terminal should be able to display the same images. The full ASCII 96 character set should be supported.

3.1.7 Tape Unit

REVS currently prepares CALCOMP plot files on magnetic tape for offline plotter input. A low speed tape unit is sufficient for this purpose and would also meet unspecified general utility needs for an STE.

3.1.8 Line Printer

Currently, at TRW Huntsville, a 300 to 400 lines/minutes printer is used. While this is a practical maximum for use over a 4800 baud line, it is a bottleneck when multiple copies of longer output listings are necessary. An on-site STE line printer should be 600 lines/minute minimum. It should support the ASCII 96 character set to meet future development needs, but at least the 64 character set is mandatory.

3.1.9 External Communication

The STE should support dial-up communication from remote sites, using RS-232-C interfaces, with selectable line transmission rates up to at least 4800 baud. The STE should support at least six input lines, which can be divided in any manner between local and remote use, and be easily reconfigured. This capability will be sufficient for all three tools, REVS, PERCAM, and AUTOIDEF.

3.1.10 Internal I/O Transfer

I/O transfer rates sufficient for effective REVS operation must be assessed in the context of CPU speed, primary memory size, and memory management strategy for each STE candidate machine. As discussed in 3.1.2, limitations on primary memory can have a profound impact on I/O requirements.

3.2 PERCAM RESOURCE REQUIREMENTS

The needs of PERCAM are generally small compared to those of REVS and AUTOIDEF. The exception may be in CPU speed where slower speeds will place limits on Monte Carlo replication capabilities.

3.2.1 CPU Speed

As discussed in Paragraph 4.2, the execution time for a PERCAM run, for a given CPU speed, is a multiplicative function of the number of "attackers", number of "defenders", number of time steps, and complexity of the ELT logic.

For single deterministic cases, typical execution times are modest (on the order of 20 to 60 CPU seconds on the CDC 7600 and 1 to 5 minutes on the CDC 6600). In the usual case of one or two runs per day, a 50:1 slowdown ratio versus the CDC 7600 would be tolerable. However, for studies requiring up to 100 Monte Carlo replications, submitted at the rate of 5 to 10 per day minimum, a slowdown of <10:1 would be desirable. For a given STE CPU speed, one must accept limits on the scenario size and number of replications. Determination of an "acceptable" CPU speed requires detailed analysis of the modeling needs of the particular user. For the STE, this can be gained only through experience with PERCAM models of typical STE applications.

3.2.2 Primary Memory

PERCAM typically executes in 80 to 100 Kbytes of memory. Working file space is less than 0.4 Mbytes. Thus, primary memory requirements are not significant compared to REVS.

3.2.3 Mass Storage

The standard PERCAM software (Library Builder, Preprocessor, Component Library, and Post-processor) occupies less than 0.3 Mbytes. Graphics software occupies about 0.05 Mbytes. Working files are usually processed and not saved. Therefore, 1 to 2 Mbytes allowance for temporary storage is generous, and is insignificant with respect to total STE requirements.

3.2.4 Graphics Hardware

PERCAM needs can be satisfied within the capabilities needed for either REVS or AUTOIDEF.

3.2.5 Plotter Hardware

A CALCOMP compatible plotter producing 8-1/2 by 11 inch plots (simple bar charts, histograms, and curves) is adequate for PERCAM use.

3.2.6 Card Reader/Tape Unit/Line Printer

PERCAM input/output requirements are minimal and can easily be satisfied within the REVS requirements.

3.3 AUTOIDEF RESOURCE REQUIREMENTS

The requirements herein are tentative, based upon preliminary information, and subject to change downward. As of this writing, Boeing Computer Services is conducting a performance improvement program to increase the speed and efficiency of the prototype system. Further information and impacts of Build 2 will be noted in the STE Final Report.

3.3.1 CPU Speed

Because AUTOIDEF is an interactive input system, CPU speed is but one component of total system performance. A basic unit of accomplishment is

the construction of an IDEF diagram. On the current Build 1 system, this takes from 20 minutes to 1.5 hours wall clock time. Average CPU time on the Cyber 175 is about 60 seconds. The rough average ratio of wall clock time/CPU time is about 50:1, indicating that CPU speed is not a bottleneck on the present system. However, the overall system performance is slower than desirable.

We would expect some performance improvements to concentrate on the DBMS software. From our experience with a similar DBMS on REVS, we found that speed can be increased up to 100:1, but at a sacrifice of portability and generality. Of greater importance is the current use by requirement of the Tektronix 4014 storage tube display terminal. Because storage tube displays cannot be selectively refreshed, the entire display must be re-painted each time anything is changed. This slows the performance of the system.

We believe that AUTOIDEF could tolerate at least a 5:1 slowdown relative to the Cyber 175 (i.e., about 30:1 relative to a CDC 7600), and perhaps more, provided software and display efficiencies are suitably increased. Analysis of the tolerable slowdown must be done for each candidate STE configuration.

3.3.2 Primary Memory

Present estimates for the Cyber 175 configuration are 170000₈ CDC 60-bit words for program space and 30000₈ words for data area. This would imply a minimum of 0.6 Mbytes of primary storage must be available to the user.

3.3.3 Mass Storage

The current data we have indicates a 50 Mword disk allowance for a large manufacturing environment project. This translates to about 400 Mbytes. AUTOIDEF presumes that all previous versions of material in the data base will be retained. A more appropriate allowance for the STE might be 5 to 10 Mbytes per "project" with an allowance for four active projects and 10 archived projects. Approximately 115 Mbytes should be sufficient for user files, source code storage and object code storage.

3.3.4 Graphics Hardware

Currently, AUTOIDEF is required to support the Tektronix 4014-1 Graphics Display Unit (storage tube type) with input via the Tektronix 4953 Graphics Tablet. This is to be compatible with current CYBERNET equipment. Dependencies on the Tektronix 4014 are limited to the user interface functions of AUTOIDEF. They are not unique to the 4014, but apply to any graphics terminal configured with a storage tube. BCS has made design provisions for eventual transfer to a refresh type terminal.

The functions that depend on a storage tube are of two types -- both concerned with the user input method. The first is the display of the menu options and the necessity for redisplaying the entire screen when the menu area is full. Secondly, input of box name and number of small boxes are entered in the lower area of the screen to avoid the extraneous information necessary to prompt for the text in the IDEF form message field.

More general requirements imposed on AUTOIDEF as stated as follows:

- "It is desirable that the IDEF tool provide a graphics data base resolution (i.e., the minimum number of addressable points) of at least 32,768 points in each direction. The graphics display must have a resolution of at least 1024 points over 15 inches on a single axis."
- "The IDEF form shall be displayed at its actual size or reasonably close. The full size of the form is 10.0 inches wide and 7.0 inches high. The message field for the IDEF diagram is 6.0 inches by 10.0 inches. The clerical information at the top and bottom of the IDEF form should be displayed (10.0 inches wide by 0.7 inches high at the top, 10.0 inches wide by 0.4 inches high at the bottom)."

Relaxation of these requirements may be possible in the STE. There is no current AUTOIDEF requirement for color display.

3.3.5 Plotter Hardware

AUTOIDEF is currently supported by a CALCOMP plotter. Any equivalent plotter supporting basic CALCOMP software interfaces and producing 8-1/2 x 11 inch plots of equal quality would suffice. Off-line support via tape input is satisfactory.

3.3.6 Card Reader/Tape Unit/Line Printer

As AUTOIDEF is basically a graphics input/output tool, there are no specific requirements for other peripherals. However, these may have a role in installation and maintenance, and a tape unit would be needed for off-line plotting. Any of these peripheral needs are satisfied within the REVS hardware requirements.

3.4 STE HARDWARE SELECTION CONSIDERATIONS

In the future evaluation of candidate machines to meet STE needs, it is important to remember that maximum performance is not the goal. The objective is to define the most economical and balanced system that can handle the anticipated user load effectively during the R&D phase, and be later

deployed for production use. In the case of current ARPANET machines, availability (i.e., percent capability effectively open for STE use) is also a major consideration.

In later operational use, the selected configuration must be accessible by users and responsive within their schedule requirements. Hence, the projected user load profiles and STE time-sharing policies are of primary importance in configuring the STE.

For the R&D phase, it will be assumed that four different groups will be accessing the STE on a regular basis. These will be a tool development contractor, a tool evaluation contractor, RADC technical specialists, and other Air Force potential users. Over these groups it is expected that no more than six individuals, and usually no more than three will be trying to use the prototype STE at a given time.

The probabilistic workload demanded from the STE is to be synthesized using the tool utilization data in Section 4 with consideration for the particular parameters of each candidate STE configuration. Then other factors of that configuration such as cost and availability are to be weighed against utilization to determine the merits of the candidate.

4.0 TOOL UTILIZATION PROFILES

This section presents typical observed utilization information about REVS, PERCAM, and AUTOIDEF to be used during the evaluation of STE candidate configurations. Because AUTOIDEF has been introduced only recently, operational information has not yet been accumulated and can only be postulated.

4.1 REVS UTILIZATION

Because REVS is a complex software system, three phases of operation must be addressed: development, installation, and production. The development phase is typical of most experimental software projects. Various alternative solutions to a problem may be tested on a trial basis, or a specific set of modifications may be implemented according to a formal schedule. Without knowledge of the specific task to be done, one cannot predict a specific activity load.

At some point, however, REVS must be installed on the host system. The same procedure is followed whether it is an initial installation or introduction of a modified version. A minimum schedule for the installation process is five working days because intermediate outputs must be inspected and verified at each step. Table 4.1 presents the typical sequence of runs, by day, and tabulates relevant parameters for each run. The schedule may be extended and re-runs made necessary by problems encountered along the way. It is expected that the installation sequence will be performed infrequently at the STE.

The "production phase" (i.e., application by users) will be the primary STE operating mode. A REVS data base may be constructed over a one to six month period, depending on its scope and complexity. The time-span of a REVS project is doubled when full simulation is employed.

Figures 4-1 through 4-4 present REVS usage data gathered in March and April 1980 from a real IV&V project employing REVS to verify software requirements for a major Army missile system. The data presented is for runs associated with the airborne computer program requirements, and represents a typical slice of project activity.

Figure 4-1 shows the distribution of days on which runs were made over a nine-week interval, and notes the number of runs each day and total CPU time used. Individual run times varied from four to twenty-seven seconds. The data base grew from an initial 224 Kbytes on the first run date to about 472 Kbytes at the end of the period.

From Figure 4-2, one can determine the relative frequencies of various numbers of runs per day over the period. Figure 4-3 enables an estimation of the number of days to the next run day (e.g., 0 means multiple runs on the same day, 1 means approximately one day between runs, etc.). Figure 4-4 shows the distribution of runs by time of day. There is significantly more activity toward the end of the day, partly to submit jobs for overnight processing so that results are available in the morning. The STE evaluation should consider such skews in loading.

Table 4.1 REVS Installation Runs

DAY	RUN NO.	DESCRIPTION	CDC 7600 CPU SEC.	I/O MWORDS	PRINTOUT PAGES
1	1	COPY PERMFILES	18.5	2.17	10
	2	LIST COMPILER WRITING SYSTEM	4.5	.57	250
	3	LIST DBCS PROGRAM LIBRARY	4.0	.44	200
	4	LIST PASCAL COMPILER	5.7	.67	300
	5	LIST REVS	21.5	2.74	1200
	6	LIST REVS TEST CASES	6.8	.89	400
2	7	CREATE RSL TRANSLATOR	29.4	2.18	100
	8	CREATE DATA BASE LIBRARY	27.1	1.10	500
3	9	CREATE REVS LOAD MODULE (COMPILE REVS)	119.8	2.60	50
	10	CREATE VVLIBE, VVDBLDR	2.2	.04	20
	11	CREATE NULL ASSM	0.6	.02	50
4	12	CREATE RISF	0.3	.01	10
	13	CREATE JSL EMULATORS	19.5	.21	20
	14	CREATE REVSLIB	5.3	.13	5
	15	NOMINAL INITIAL ASSM	4.6	.07	25
5	16	CONSTRUCT PASCAL LIBRARY (PLIB)	22.3	.48	50
	17	MERGE PLIB AND REVSLIB	7.4	.22	30
	18*	EXECUTE TEST CASES	~304.0	--	--
		*ACTUALLY UP TO 35 SEPARATE JOBS			
		TOTALS	~604	>14.54	>3220

RAD80-017

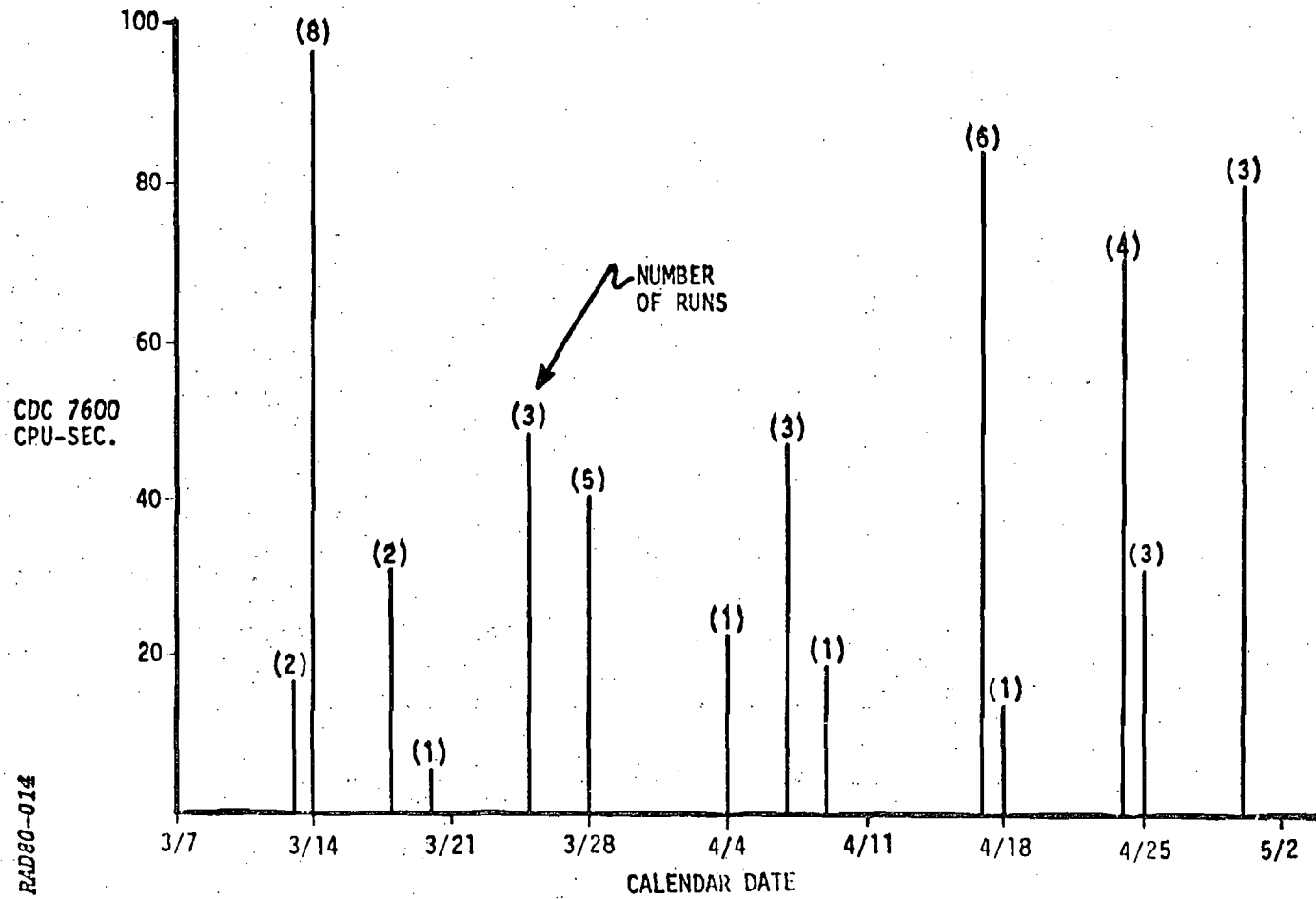


Figure 4-1 Segment of REVS Project Run History

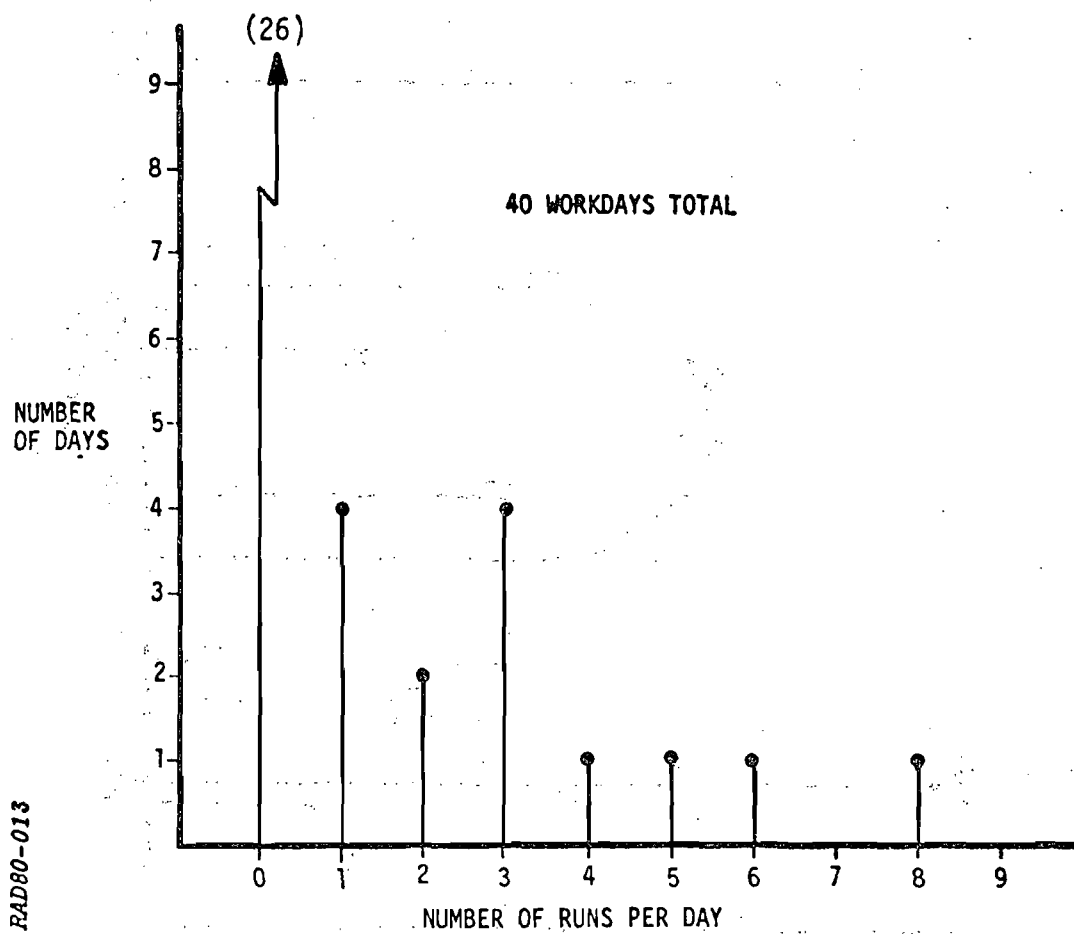


Figure 4-2 REVS Runs/Day Frequencies

PAD80-012

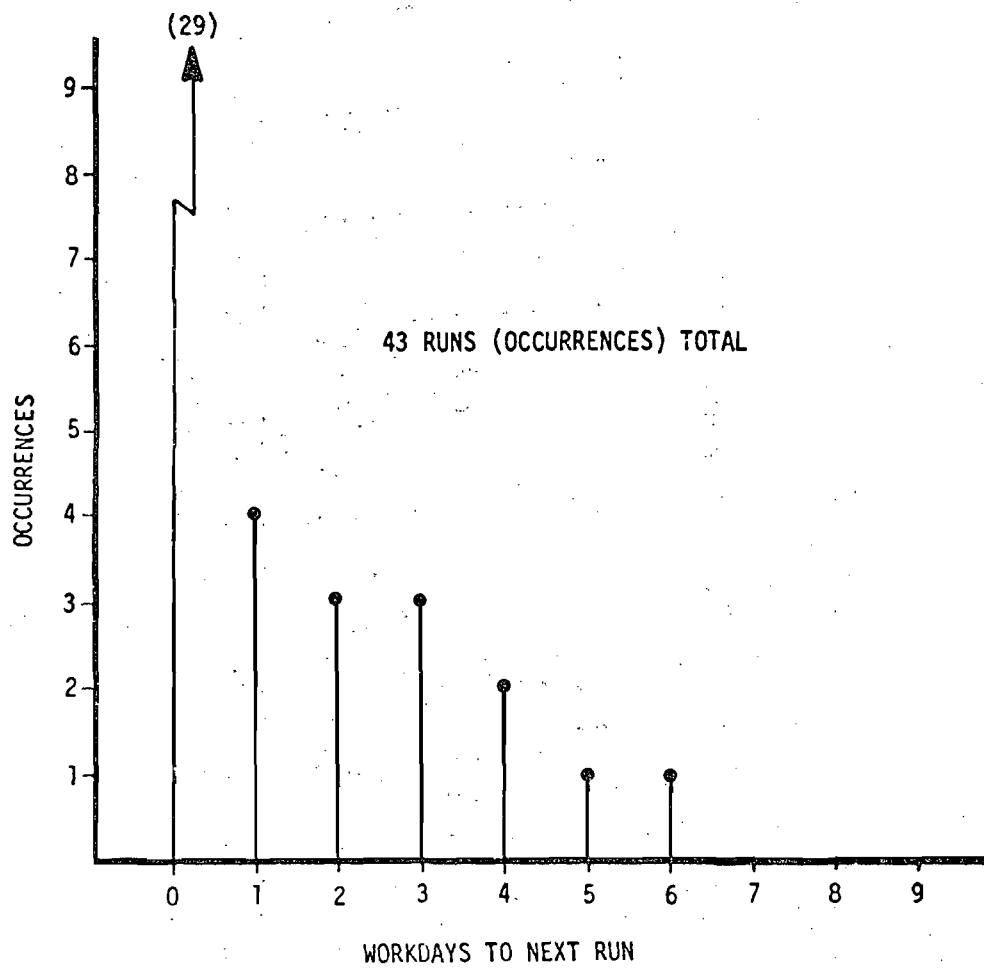


Figure 4-3 Intervals Between REVS Runs

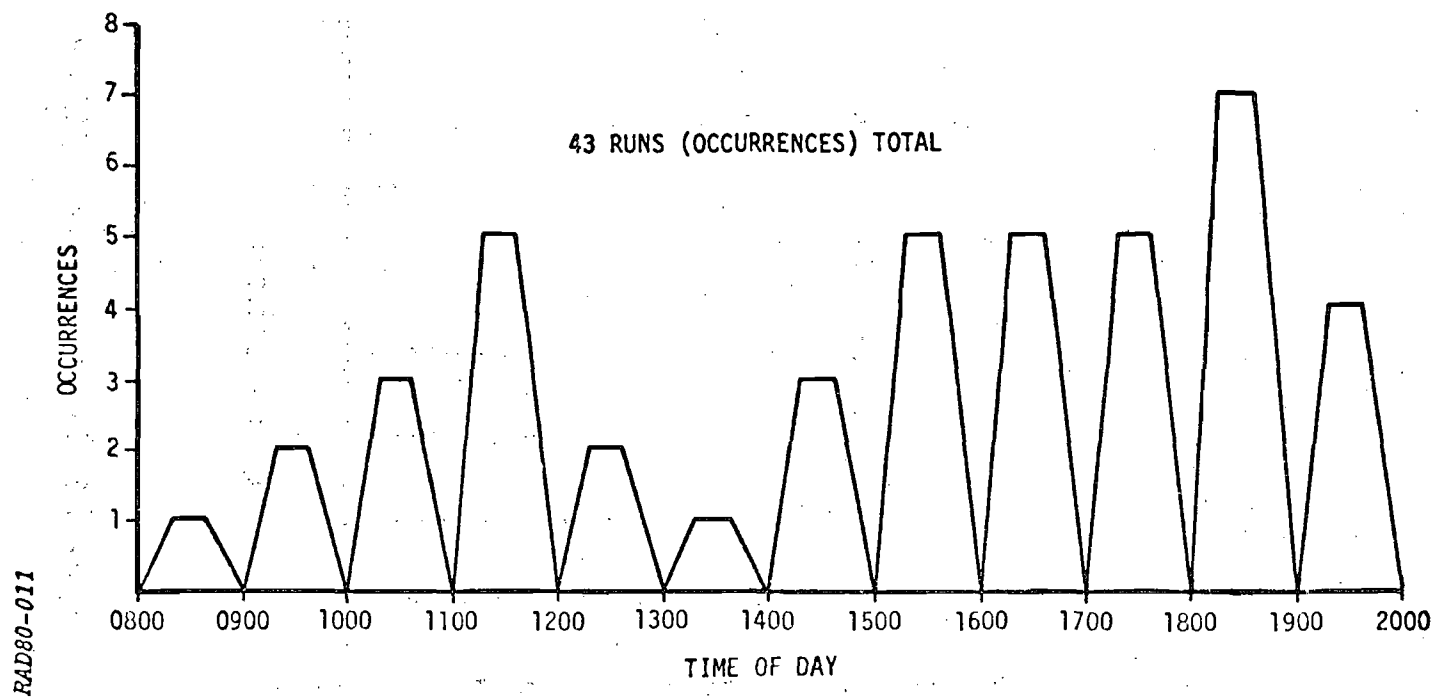


Figure 4-4 Time of Day of REVS Runs

4.2 PERCAM UTILIZATION

The time-span of PERCAM projects varies from two weeks (narrowly-focused efforts) to a year (complex tactical scenarios and issues). The typical small project employs a deterministic model to examine the basic relationships and function of a conflict system. Larger and more complex projects involve the statistical simulation of systems with emphasis on sensitivity analysis and more detailed modeling. PERCAM is used with a "learning curve" philosophy. A simple model is initially built and then progressively corrected, refined, and expanded as understanding of the problem is gained. For most projects, a consistent one or two runs per day are required.

Table 4.2 illustrates a typical project run history for a two man-week project. The first week was spent in learning the basic model and tuning it to simulate the required constraints. The second week was spent in completing the model to display the engagements that were desired. As shown by the table, there is a growth in the amount of computer program time required as the model reaches its maturity, and then a final set of runs as the completed model is executed for final results. Table 4.3 illustrates the history of a longer effort that passed through three distinct phases.

Table 4.2 Example PERCAM Study Number One

<ul style="list-style-type: none">• 20 ATTACKERS VERSUS ONE DEFENDER• TWO MAN-WEEK EFFORT<ul style="list-style-type: none">- ONE WEEK FOR THE LEARNING CURVE AND EVENT LOGIC TREES- ONE WEEK FOR COMPUTER RUNS AND DOCUMENTATION• CDC 7600 COMPUTER UTILIZATION - 270 SECONDS			
<u>CHECKOUT</u>		<u>STUDY</u>	
<u>RUN NO.</u>	<u>CPU SEC.</u>	<u>RUN NO.</u>	<u>CPU SEC.</u>
1	11.445	8	22.210
2	20.161	9	22.080
3	20.133	10	22.005
4	20.720	11	22.023
5	22.019	12	21.722
6	21.199	13	21.687
7	22.582		
SUBOTAL	138.259	SUBTOTAL	131.727

RAD80-018

The execution time of a PERCAM model is a multiplicative function of the following factors;

- Number of time steps over the simulation interval
- Number of "attackers"

Table 4.3 Example PERCAM Study Number Two

- 18 ATTACKERS VERSUS 20 DEFENDERS
- FOUR MAN-WEEK EFFORT
 - TWO WEEKS FOR MODULE BUILDING
 - ONE WEEK FOR MODEL VERIFICATION
 - ONE WEEK FOR PARAMETRIC AND SENSITIVITY ANALYSIS
- CDC 7600 CPU UTILIZATION - 457 SECONDS

MODEL BUILDING PHASE		MODEL VERIFICATION PHASE		PARAMETRIC AND SENSITIVITY ANALYSIS PHASE	
RUN NO.	CPU SEC.	RUN NO.	CPU SEC.	RUN NO.	CPU SEC.
1	8.492	11	22.219	17	50.271
2	10.318	12	20.719	18	41.235
3	16.191	13	20.917	19	41.404
4	16.382	14	11.911	20	42.746
5	4.599	15	12.675		
6	18.918	16	22.435	SUBTOTAL	175.656
7	20.567				
8	20.381	SUBTOTAL	110.876		
9	16.701				
10	38.583				
SUBTOTAL	170.072				

RAD80-019

- Number of "defenders"
- Number of Monte Carlo replications
- Complexity of the Event Logic Trees.

The amount of memory required is an additive function of the complexity of the executive and the complexity of the Event Logic Trees plus a multiplicative function of the following:

- Number of attackers
- Number of defenders
- Number of attacker/defender parameters.

Typically, when Monte Carlo runs are needed, jobs are submitted in small increments over several days, then subjected to final post-processing.

4.3 AUTOIDEF UTILIZATION

Construction of a single IDEF diagram requires 20 to 90 minutes. A diagram contains up to six "boxes", each of which may be detailed in a lower level diagram. A full day's work would involve construction of a diagram and the diagrams for its subordinate boxes. More experience needs to be accumulated before a typical pattern of activity is apparent. Further performance improvement efforts are in progress.

5.0 STE REQUIREMENTS SUMMARY

This section consolidates requirements for the STE in an abbreviated checklist form, considering the combined needs for REVS, PERCAM, and AUTOIDEF. Software support requirements are listed in Paragraph 5.1 and hardware requirements are listed in Paragraph 5.2. In both of these paragraphs, the tool or tools that primarily drive each requirement are indicated in parentheses -- (R) for REVS, (P) for PERCAM, (A) for AUTOIDEF (or combinations of these). Other requirements, without identifiers in parentheses, are inserted to provide features necessary to support an STE environment independent of the specific tools. Integration issues identified in the compilation of these requirements are briefly discussed in Paragraph 5.3.

5.1 SOFTWARE ENVIRONMENT REQUIREMENTS SUMMARY

- Pascal Compiler (R)
 - extensions per Paragraph 2.1.6.1
- FORTRAN Compiler
 - ANSI FORTRAN 66 with following extensions:
 1. random file I/O (R,A)
 2. MASK, SHIFT, ENCODE, DECODE functions (R,A)
 3. support of '><' characters (P)
 4. Namelist I/O (P)
 5. operand conflicts allowed in arithmetic statements (A)
 6. use of literals instead of Hollerith data allowed (A)
 7. use of ENTRY statements allowed (A)
- Pascal application to operating system communication capability per Paragraph 2.1.6.2 (R)
- ASCII 96 character set desirable, 64 character set minimum (R,A)
- Memory Management (R,A)
 - segmented loader transparent to application program (or)
 - virtual memory management system
- Plotter Support Software (R,P,A)
 - CALCOMP FORTRAN interfaces
 - equivalent to basic CALCOMP routines
- Graphics Support Software
 - FORTRAN callable (R,A)
 - compatible with ACM/SIGGRAPH GSPC Core System (A)

- Job Control
 - multi-phase (R,P)
 - Modifiable from within application program (R)
- Dynamic file access (R,A)
- Possible compatibility with check-point restart capability at later date (P)
- Library management utility (R,P,A)
- Sort/merge utility (P)
- Independent data entry/edit utility
- Time-sharing environment support
- Local/remote communications support

5.2 HARDWARE ENVIRONMENT SUMMARY

- CPU speed (R,P,A)
 - only machines with less than 55:1 slowdown relative to CDC 7600 will be considered
- Primary Storage (R)
 - more than 1 Mbytes; 2 Mbyte nominal
- Mass Storage (R,P,A)
 - at least 187 Mbytes on-line plus (to be determined) allowance for system, user I/O files
 - (or) at least 120 Mbytes on-line with archives on removable disk packs
- REVS Color Graphics Hardware (R,P)
 - requirements per Paragraph 3.1.4
- AUTOIDEF Graphics Hardware (A)
 - requirements per Paragraph 3.3.4
- Plotter Hardware (R,P,A)
 - CALCOMP compatible, 8-1/2 by 11 format minimum
 - continuous-feed 12 inch and 30 inch paper widths desirable for REVS, if available
 - off-line operation, tape input
- Data Entry Devices (R,P)
 - card reader and/or alphanumeric terminal as desired by STE -- tools set no particular requirements except ability to accept 80 column card images (and display them in the case of CRT terminals).

- Tape Unit (R,P,A)
 - low speed utility acceptable
 - used primarily to generate plotter tapes
- Line Printer (R)
 - 600 lines/minute minimum
 - ASCII 96 character set desirable, 64 character minimum
- Internal I/O Transfer Rates
 - to be assessed for each configuration in conjunction with CPU speed, memory configuration

5.3 STE INTEGRATION ISSUES

The requirements levied on the STE by the tools form a compatible set except for the graphics hardware requirements of REVS and AUTOIDEF. REVS uses a color graphics capability, while AUTOIDEF currently uses a Tektronix 4014 high resolution, monochrome, storage tube device of the type used for engineering drawing applications. Use of a monochrome display for REVS is feasible, but not desirable for human engineering reasons. Color is valuable for readily differentiating different types of display nodes, menu selections, and error conditions. Sufficient size and resolution may be obtainable within the REVS requirements to support AUTOIDEF needs for the STE. During future evaluation of graphics equipment, we will assess means of adequately meeting the needs of both tools with a single terminal.

Both REVS and AUTOIDEF use variants of the University of Michigan ADBMS data base management system. It may be feasible to satisfy the needs of both tools with a common system, although one or both of the tools may need to be modified. This possibility should be explored during future implementation of an STE.

6.0 SURVEY OF ARPANET SYSTEM HOSTS

This section presents the results of Task 2 of the STE Study. The objective of Task 2 was to determine existing computer systems accessible through the ARPANET which have sufficient resources available to provide the STE and which meet the requirements defined by Task 1. Further objectives were to obtain costs for using the system and availability over the next five years, and to identify systems which could be improved sufficiently to provide the STE.

We have identified several ARPANET nodes, using CDC and non-CDC machines, that have the capability to support the STE. However, reliable data on costs and availability proved nearly impossible to gather, and was not known to the ARPANET Network Information Center. Two nodes, Lawrence Berkeley Labs (LBL) and Argonne National Labs (ANL) readily cooperated to provide as much information as possible, but other nodes were reluctant about making projections.

Costs of using REVS were estimated using data provided by LBL and ANL, and these estimates were comparable. However, comparison of these estimates with actual charges for REVS runs at the Naval Air Development Center (NADC) during installation in May 1979 indicated that the NADC charges were approximately fifteen times the estimates for LBL and ANL. The cause of this discrepancy could not be ascertained. Therefore, the NADC charges and LBL/ANL estimates should be regarded as "high" and "low" boundaries on costs.

Similar problems were encountered in trying to determine accurate slowdown ratios between various families of computer systems and the CDC 7600. We have accurate comparisons between CDC 7600 and 6600 series systems, based on actual REVS runs that shows an average 6600 slowdown ratio of 5.7:1 relative to the 7600. However, no source of reliable information comparing computers from different vendors could be found. Standard sources (e.g., Data Pro's EDP Buyer's Bible, Auerbach Buyer's Guide) do not provide a basis for comparison between manufacturers, and other investigators have found inaccuracies in what data are provided [9]. We have, thus, had to rely on estimates provided by TRW personnel experienced on several machines.

Paragraph 6.1 discusses methods of investigation used in Task 2. Paragraph 6.2 presents information gathered on CDC ARPANET hosts, while Paragraph 6.3 presents information on non-CDC ARPANET hosts. In these paragraphs, nodes are referenced by their acronyms for brevity (e.g., LBL for Lawrence Berkeley Labs). The full name of each node, and relevant node configuration information can be found in Appendix A, which is ordered by acronym. Paragraph 6.4 considers the Honeywell 6180 hosts and rejects them for lack of a suitable Pascal compiler on Honeywell machines.

Paragraph 6.5 presents available data on REVS running costs at NADC (an existing installation). Paragraph 6.6 discusses our conclusions about ARPANET hosts for the STE.

6.1 METHODS OF INVESTIGATION: TASK 2

The following topics were investigated for each of a number of initially screened ARPANET host sites:

- Compliance with STE Software/Hardware Environment Requirements (summarized in Section 5).
- Job Scheduling, Billing Algorithms and their Parameters.
- Individual/Combined Host Utilization.

Data were gathered by querying the ARPANET Network Information Center's on-line database, by telephone interviews with candidate host liaisons, and by ARPANET mail communications. Information regarding job scheduling and cost accounting algorithms was gathered by telephone interviews with each candidate host's ARPANET liaison, software systems personnel, and from documentation on the candidate host's operating system.

ARPANET host utilization data was provided by certain ARPANET host liaisons. The data reflected two measures of host utilization: First, we were concerned to measure the host operating system load irrespective of the source of jobs (local batch, ARPANET interactive, etc.). On this score, data was primarily qualitative and provided in the form of "educated guesses" as to the average percent of total host capacity utilized on an hourly and monthly basis. Secondly, we were concerned to measure host utilization originating in ARPANET activity (file transfer, interactive, mail, etc.). The available data were presented in the form of packets transferred to a given host per month. Regrettably, statistics distinguishing interactive, mail and file-transfer activity at candidate ARPANET hosts were not available.

6.2 CDC ARPANET HOSTS

In this paragraph, we present those CDC ARPANET system hosts which are adequate to each tool in the STE and to the STE as a whole. The tools comprising the STE were originally developed on Control Data machines and operating systems. We indicate those CDC ARPANET system hosts which use the "baseline" operating system for a given STE tool in the following discussion.

6.2.1 Hardware/Software Functional Requirements

Table 6.1 presents the status of each CDC ARPANET system host configuration as determined by the hardware/software functional requirements imposed by the tools of the STE.

Table 6.1 CDC ARPANET Host Evaluations

	GOOD	MARGINAL	UNACCEPTABLE	UNDETERMINED
PERCAM	LBL, EGLIN, AFWL, NADC, DTNSRDC, BNL, NSWC-WO, NSWC-DL, AFWL, FNBC			NYU, WPAFB
AUTOIDEF	LBL, EGLIN, AFWL, NADC, DTNSRDC, BNL, NSWC-WO, NSWC-DL, AFWL, FNBC			NYU, WPAFB
RSL/REVS	LBL, BNL, NADC	EGLIN, AFWL, DTNSRDC, NSWC-WO, NSWC-DL	FNBC	NYU, WPAFB

PERCAM was originally developed under the CDC SCOPE operating system, AUTOIDEF under the CDC NOS operating system, and RSL/REVS under the CDC SCOPE operating system (with subsequent modification and transport to a KRONOS operating system at NADC).

Both the KRONOS and NOS operating systems run on various machines at NADC which makes this host a good installation and integration site for the STE. To date, REVS and PERCAM have been installed at NADC, but AUTOIDEF has not been installed.

The NOS/BE operating system at EGLIN would provide an adequate operating environment for STE installation and integration since it extends NOS with certain KRONOS capabilities. There is the caveat, however, that the Pascal compiler at EGLIN has uncertain origins. It resides in a user file but not in a system file and, thus, receives little or no system maintenance. The compiler installed with REVS at NADC could possibly be adapted, but the extent of required modifications cannot be estimated precisely without detailed comparison of the NADC and EGLIN operating systems. Also, the job control language interfaces with REVS must be verified to be compatible with current installations. Considerable variation exists between different CDC operating systems and site versions.

SCOPE, widely regarded as a "friendly" operating environment for large scale CDC software transfers, runs at AFWL, BNL, and NSWC-DL. However, AFWL and NSWC-DL are rated marginally adequate for STE installation and integration because their primary memory configurations are less than generous. The principle concern would be with REVS operations which might well tax the memory resources at these hosts, especially interactive operations during peak system load periods. If the primary memory configurations at these hosts were upgraded to meet the requirements of REVS operations, they would provide adequate sites for STE installation and integration.

The BKY and SESAME operating systems at LBL are based on SCOPE version 1.6 with extensions made to suit local service requirements. All other factors being equal (an unusually optimistic assumption about computer operations), LBL would be a good operating environment for STE installation and integration since BKY and SESAME use the same program interfaces as do the other CDC operating systems. LBL personnel have been most helpful. They alone, among CDC ARPANET personnel, provided us with data on host utilization, job scheduling and job costing. Moreover, LBL's hardware configuration is similar to those encountered in past STE installation and application experiences. We can, therefore, offer our assessment of LBL's STE-adequacy with more assurance than in the case of other CDC ARPANET system hosts. Once again, detailed operating system comparisons must be made before a precise estimate of required modifications can be made.

FNWC is rated unacceptable because the system software at that host lacks a Pascal compiler. Apart from this deficiency, FNWC would provide a good environment for the STE since it runs the SCOPE operating system and has a good memory configuration for STE installation and integration. Although the current compiler used with REVS could probably be installed with little difficulty, FNWC is apparently reluctant to allow such installations.

In sum, LBL, BNL, and NADC appear to be the best candidate CDC ARPANET hosts to satisfy the STE hardware/software functional requirements.

6.2.2 CDC Host Job Scheduling Considerations

All CDC ARPANET host operating systems unlike, say, UNIX or MULTICS, are primarily batch oriented. That is, incoming jobs are delayed in an input queue for service and are scheduled for execution on the basis of resource requests (e.g., CPU time, memory, peripherals) and a user-supplied priority. The priority of a job may be modified downward by the job scheduler if allocating the requested resources would significantly reduce system throughput. Jobs are initiated by entry into an "active" queue and multiplexed in a round-robin fashion for time quanta that vary so as to enhance system throughput. Interactive jobs are treated as essentially batch jobs with the highest priority and immediate entry into the active queue as default characteristics.

The similarity among CDC ARPANET host job scheduling policies is a vestige of their common origin out of an earlier CDC operating system. The differences that do exist in job scheduling policies across CDC ARPANET hosts attach primarily to the weight given the various resource quanta that affect a job's priority but not to the strategy of selecting the order in which jobs are executed.

For instance, at LBL, the key factor (total job "computing units" (CUs), see Section 6.2.4.1 below) determining a job's effective scheduling priority is computed as a weighted sum of its requested CPU time, memory and peripherals service. The critical breakpoint value according to user service contacts is 63 CUs.

In the next paragraph, we detail the job scheduling algorithms and policies at LBL, thus providing a paradigm case of CDC ARPANET host job scheduling.

6.2.2.1 Job Scheduling at LBL

Job resources include job slot identifiers called "control points", small and large core memory (SCM and LCM), the CPU, tape drives, data cells, the chip store, and unit record devices (printers, punches, microfilm processor, Calcomp plotters).

Resources are allocated on the basis of an "urgency" factor which is a function of jobcard priority, the age of a job (its time in the system), and other factors turning on the resource being scheduled. The aging factor is cumulative through the life of the job and increases at a rate dependent on the jobcard priority.

Three general types of scheduling are encountered as a job flows through the system -- job initiation, resource allocation during execution, and queued file processing.

6.2.2.1.1 Job Initiation at LBL

A job enters the system when it reaches a 6000 (the B (6600) or C (6500) machine). Jobs are then sent to the specified machine to be placed in the input queue. Rush and normal jobs with a CU limit of 63 or less will have a higher urgency for initiation than other rush jobs. This is done to allow fast turnaround for debug jobs.

Jobs destined for the 7600 must first pass through the common 6000/7000 input queue. While in the input queue, a job's urgency is expressed as a two digit octal number, CA where C is the job class (determined by jobcard priority and whether the account number is DOE funded or not), and A is the age of the job (determined by the time in the system). This two digit number, urgency, is used to determine the order in which jobs are sent on to the 7600. The same general scheme is used for 6000 jobs. Once the job's urgency is determined (by the method described above), the 6000 scheduler can determine which jobs to initiate.

The C parameter used in determining urgency is determined from Table 6.2. The A parameter is assigned on the basis of the maximum number of hours the job has been in the system, as shown in Table 6.3.

6.2.2.1.2 Resource Allocation During Execution at LBL

Each executing job in the 7600 and 6000s is assigned a job slot identifier called a control point. The 7600 has 127 control points and the 6000 system has 63. This limits the total number of jobs that may be executing at any one time.

The scheduler attempts to optimize throughput (jobs completed per unit time) by having as many jobs as possible doing things at once. Thus, for instance, a job which is expected to start some I/O in a short while is given

Table 6.2 Input Queue Classes (C Parameter)

JOB CARD PRIORITY	NON-DOE FUNDED	DOE FUNDED
DEFERRED (2-4)	1	2
NORMAL (5-7)	3	4
RUSH (10-16)	5	6
INSTANT (5-16 & <= 63 CUs)	7	7

Table 6.3 Age Computation for Input Queue (A Parameter)

MAXIMUM HOURS IN SYSTEM	A
LESS THAN 1	0
1	1
4	2
10	3
16	4
38	5
52	6
INFINITY	7

the CPU in preference to a job which is expected to compute for a long time. Then the first job can do I/O while the other is using the CPU. This tends to decrease the turnaround time slightly for the CPU-bound job, but it increases the system throughput dramatically. The guiding principle, again, is this: Maximize throughput while allowing for special cases.

The 7600 and the 6000s basically schedule memory occupancy and CPU use. Jobs in main memory that are ready for execution may be rolled out to disk if a job of higher urgency needs the memory space. Jobs waiting for some device or staging also may be rolled out to disk. The scheduler queues them for roll-in when they are ready to run and their urgency is sufficiently high.

The urgency for LCM occupancy (see scheduling formula below) includes a factor for computing units (CUs) remaining until the CU limit. The job with the highest urgency (the primary job) is the one put in LCM first. If the job in SCM needs to reference the disk to reload or empty an LCM buffer, it is swapped out to LCM. Rollout to disk occurs when a job attempts to stage, or when another job has a higher urgency for LCM occupancy. Job initiation occurs when a job in the input queue has a higher urgency than one of those in LCM or when there is no executing job which is ready to use the CPU.

The factors in the urgency for CPU use are CPU burst time, CUs remaining to CU limit, and field length. The last two factors are relatively unimportant. Urgency for roll in/out includes factors for devices attached and field length where lengths of 160,000 bytes are used in considering field length.

Executing interactive jobs receive special urgency considerations whenever they are in any queue.

The operators can increase or decrease an executing job's urgency. This is done by an entry at the operator console, and is subject only to the operator's discretion. The phrase used to describe this is "forcing a job".

The length of the file is a primary factor in the urgency for processing queued files. Secondary to length is the jobcard priority and the age factor of the job. Only when the job has been in the queue for a long time will the priority or age factor over-ride the length factor. The order of processing is always subject to operator intervention.

6.2.2.1.3 Scheduling Algorithms at LBL

On the 7600, there may be six jobs in LCM at once. The scheduling priority, P (or -- the urgency for LCM occupancy) of a job is a single integer calculated as follows:

$$P = K + \text{FORCES} + D + B \cdot \text{AGE} - E \cdot (\text{LOG}_2(\text{CUR})) + (\text{DOE BONUS}) \\ - (\text{DISK PENALTY}) - (\text{LCM PENALTY}) + (\text{RUNNING BONUS}) \\ + (\text{DISK ALLOCATION BONUS}) + (\text{INSTANT BONUS})$$

Each job belongs to one of the following four classes -- rush, normal, deferred, and background. Of the various parameters in the algorithm, only A, B, and C are class dependent. All the terms above are expressed in terms of an effective age (in minutes). The priority for a job is essentially that age at which its effective age is the highest in the machine.

The following is a definition of each term in the expression for P:

K	A constant chosen to make P positive in all cases.
FORCES	... are applied by the operator or the system in order to move a job above all competition. The system automatically forces certain jobs necessary for its own integrity.
D	0 for normal jobs, 240 (4 hours) for rush jobs, -10,000 for deferred jobs (they can never compete with normal or rush jobs) and -20,000 for background jobs.
B	... is 1.2 for rush, 110 for normal, and 0.8 for deferred.

AGE	The time in minutes since the job has entered the queue. It can reach a maximum of 4095 minutes (2.5 days).
E	60 minutes for all jobs.
CUR	CUs remaining to CU limit. (NOTE: $-C*(\text{LOG}_2(\text{CUR}))$ gives shorter jobs better turnaround. Each factor of 2 in CUs is worth 60 minutes for all jobs. This LOG2 dependence, though not a fine enough mapping, provides a reasonable means to reward short jobs and at the same time not penalize large jobs excessively (as would be the case, e.g., with linear dependence).
DOE BONUS	90 minutes. Given to jobs with DOE account numbers.
DISK PENALTY	Reflects the fact that users who specify more than 10,000 sectors will cause a dramatic increase in system staging. It is 180 minutes.
LCM PENALTY	Reflects the fact that a job that uses too much LCM reduces system throughput by keeping other jobs out. The current value is $\text{MAX}[(\text{TOTAL LCM} - 600\text{K})/10\text{K}, 0]$ minutes.
RUNNING BONUS	1024 minutes awarded to a job in LCM since it need not be rolled in.
DISK ALLOCATION BONUS	180 minutes awarded a job which is using a lot of disk and, thus, "paying" the disk penalty assessed on its disk space request.
INSTANT BONUS	450 minutes awarded to a job which can be finished in about 30 seconds or less so as to reflect the efficiencies caused by the subsequent release of its resources.

The central processor is scheduled by assigning it to the least cpu-bound job available. Thus, the single highest priority job is chosen first. Then the highest priority job which will "fit" with the first is chosen, followed by the highest which fits with the other two, etc., until no more jobs can fit or the maximum of 6 jobs manageable by the system is achieved.

LCM space available will almost always be the factor limiting the number of jobs running. It turns out that maximum throughput is achieved by slightly overcommitting the CPU.

6.2.3 CDC Host Utilization Considerations

Data on this subject were the most difficult to acquire. The following general facts do hold across CDC hosts. Host utilization varies by time of year, week, and day. In late August and early September, i.e., at the end of the fiscal year, host utilization increases as users seek to exhaust their computer usage budgets. During the Thanksgiving, Christmas and other

holiday seasons, host utilization drops off. Weekly host utilization tends to be at its lowest on weekends except for system maintenance and software upgrades.

The most significant indicator of host availability as a function of host utilization is therefore the variation in system load during weekdays not falling in a holiday season or toward the end of the fiscal year. From this standpoint, we gathered the following particulars from some of the CDC hosts. Others were unwilling to release such information.

- LBL

- The Department of Energy is the preferred LBL user and is given priority over all others, deadlines notwithstanding.
- Prime time shift: 9 a.m. - 5 p.m. The host machines are operating at or near the 83% utilization maximum. Between 10 and 12 a.m., there are approximately 5 users awaiting interactive service.
- Non-prime time shifts. An average of 10% of host machine capacity is available. Interactive response is rapid and batch jobs encounter only short delays. Caveats: Large accounting and environmental measurement jobs are run between the hours of midnight and 2 a.m. and maintenance and house-keeping procedures are run on one but rarely both of the 6000 series machines at around 6 a.m. which leads to some degrading of interactive response time and some increase in batch job delays.

- NSWC-DL

- Interactive services are available only between 7:30 a.m. and 9 p.m.
- As with LBL, interactive lines are "saturated" around 11 a.m. and again around 2 p.m.
- The highest priority jobs encounter a delay of no longer than 45 minutes, the second highest priority jobs are delayed no longer than 90 minutes, and the third highest priority jobs are delayed no more than 2 hours.

6.2.4 CDC Host Job Billing Considerations

The ARPANET liaison and software systems personnel at LBL have provided us with excellent documentation on their job billing algorithms, policies and procedures. LBL's hardware configuration includes a CDC 7600, 6400 and 6600 and is therefore similar to configurations referred to in certain STE utilization studies discussed below. In what follows, we present the job accounting algorithm at LBL and apply it to the findings of the utilization studies.

6.2.4.1 LBL's Job Accounting Algorithm

Jobs run on the CDC 7600 at LBL average \$800 per real-time hour used. Those run on the 6400 and 6600 machines average \$200 per real-time hour.

These figures are fairly representative of utilization costs across CDC ARPANET system hosts.

Job billing at LBL is determined by a basic charge for system services plus an overhead rate computed on the basic charge as follows:

$$\text{TOTAL JOB COST} = (1 + \text{OVERHEAD}) * \text{AUs}$$

where OVERHEAD = 40.5% and

$$\text{TOTAL JOB AUs} = W * [J + (\text{CUs} * P)]$$

An AU (accounting unit) is \$.035, and the CUs ("computing units", see below) measure job CPU time, I/O activity, terminal connect time, memory usage, tape mounting and materials charges. P is a priority factor dependent on user specified job priority and day of the week as follows:

(priority 2 - 4, "deferred"), weekdays	.75
(priority 2 - 4, "deferred"), weekends/holidays	.50
(priority 5 - 7, "normal")	1.00
(priority 10-16, "rush")	2.00

W depends on the day of the week and target machine on which the job runs as follows:

6000s, weekdays	1.00
6000s, weekdays/holidays	.50
7600, weekdays	1.00
7600, 2 day weekends/holidays	.75
7600, 3+ day weekends	.50

J is a job initialization charge dependent on the target machine: J=10 for the 7600 and J=2 for the 6000s. NOTE: Running a job in two halves increases its overhead costs. The number of CUs per job run on the 7600 is determined by the following algorithm:

$$7600 \text{ JOB CUs} = 3 * (\text{CP} + \text{SS}) + .5 * \text{BLD} + \text{ITO} + \text{STAGING}$$

Where CP is the amount of CPU time used by non-system related activity in a job step, SS is the number of "system seconds" spent in system monitor operations, BLD is the number of large core buffer loads in the job, staging is the cost associated with job step preparations, and ITO, an "interference to others" measure, is given by the formula

$$\text{ITO} = [4 * \text{MAX}(1.2 * \text{CP}, \text{BLD}/3) - (3 * \text{CP} + \text{BLD}/2)] * \text{LCM}/4,000,000$$

For the 6400 and 6600 at LBL, we have:

$$6000 \text{ JOB CUs} = (M * \text{CP} + 20 * \text{KMR}) * (1 + \text{CM}/32768) + (10 * \text{MT}) + (5 * \text{AT}) + \text{TTY}$$

Where M=.7 for the 6600, M=.4 for the 6400 (6600 speed = 1.75*6400 speed), KMR is the number of operating system monitor requests in thousands, CM measures the "instantaneous" field length (dynamically variable), MT is the

number of tapes mounted by an operator, AT is the number of tapes mounted automatically, and TTY is the number of CUs charged for terminal connect time depending, as follows, on time of day and connection origin:

10 a.m. - 6 a.m.	1CU/connect min.	x 4 for
6 a.m. - 10 a.m., 6 p.m. - midnight	.5CU/connect min.	ARPANET
midnight - 6 a.m.	.25CU/connect min.	connection

6.2.4.2 Estimating STE Job Costs at LBL

STE utilization of CDC ARPANET hosts satisfying the hardware/software environment requirements summarized in Appendix A can be modelled directly in terms of the results of STE tool utilization studies presented in Paragraphs 4.1 and 4.2. Given the billing algorithms used at a representative CDC host such as LBL, the utilization study provides a model case of STE job cost estimation for CDC hosts with similar hardware/software configurations.

6.2.4.2.1 REVS File Transfer and Storage Costs

In this section, we discuss the costs associated with the transfer and storage of mass storage files for the different phases of REVS operations. The first cost we consider is for transferring files over the ARPANET via the File Transfer Protocol (FTP). Most ARPANET hosts do not charge for the use of an FTP connection but only for the use of interactive connections. The use of FTP channels therefore involves only those costs that result in operations on the host's file system, e.g., allocating, opening, closing files to be FTPed. These costs are of the same scale as those associated with the intended file activity for the various REVS operations. We therefore discuss the costs associated with file activity in REVS operations.

Each REVS phase makes use of several mass storage files. The largest of these will determine the costs associated with REVS file storage since the LBL policy for storage charges involves a cost of \$1.50 per 5500 word segment (program storage allocation unit, PAU) assessed on the maximum number of PAUs in use during a given month. Since the greatest REVS file activity originates in batch jobs, we may safely assume that this cost will be determined by the largest file associated with a job in a given phase of REVS operations. The associated cost for each such phase is detailed as follows:

LARGEST MASS STORAGE FILE	SIZE (WORDS)	COST MONTHLY)
Installation Phase	277,929	\$ 75.80
Production Phase	616,080	\$168.02
Application Data Base	200K - 450K	\$54.55 to \$136.00

Assuming that the production phase of REVS operations proceeds during the same month that REVS is installed, the \$168.02 charge for the largest production phase mass storage file will dominate the storage costs through that period. There are, of course, minor charges associated with mounting and entering the files into the LBL PSS library, but not of the same scale.

6.2.4.2.2 REVS Installation Job Costs

The results of the REVS utilization study indicate the running of up to 52 separate jobs during the installation phase. The following data were accumulated during REVS installation:

CDC 7600 CPU SEC.	I/O MWORDS
minimum 0.6	minimum < .01
maximum 304.0	maximum 2.74
total =604.0	total >14.54

We make two assumptions about jobs run at LBL which are borne out by representative samples. First of all, a rate of 40 BLD CUs per I/O megaword involved in a job is assumed and, secondly, a job STAGING factor of .034 times the total BLD CUs accumulated. From this standpoint, the cost of installing REVS at LBL would be:

$$\text{TOTAL (across jobs) AUs} = (1 + .405) * \text{AUs}$$

where

$$\text{AUs} = 1.00 * [52 * 10 + (\text{TOTAL JOB CUs} * 1.00)]$$

assuming for the worst case a W factor of 1.00. We then have

$$\begin{aligned}\text{TOTAL JOB CUs} &= 3 * (\text{CP} + \text{SS}) + .5 * \text{BLD} + \text{ITO} + \text{STAGING} \\ &= 18.2 \quad + 300 \quad + 1000 + 20 \\ &= 3132\end{aligned}$$

hence we have

$$\text{TOTAL AUs} = 1.405 * 3652 = 5130$$

which yields

$$\text{TOTAL INSTALLATION COST} = \$180.00$$

Note that this installation cost estimate assumes that installation phase jobs tend to be CPU bound, a favorable characteristic from the standpoint of job scheduling at LBL and, presumably, other CDC hosts. Our assumption of 40 BLDs per megaword of I/O may be over conservative. If not, tuning the I/O parameters of REVS could reduce this cost factor so as to accord with the above results. The installation estimate is based on a scenario in which the installer operates via a remote connection to LBL over the ARPANET and transfers files to and from LBL via FTP for batch job submission and local spooling of results to a lineprinter. This justifies the otherwise minimal STAGING cost associated with the interactive preparation of job streams.

6.2.4.2.3 REVS Production Phase Costs

Paragraph 4.1 details the results of a typical REVS project on which 43 jobs were run over a nine week interval. Individual runs varied from four to

twenty-seven seconds and involved an application data base growth from 224 Kbytes to 472 Kbytes. If we assume that CPU seconds account for about a fifth of the total CUs accumulated by a job, as in the installation phase, we have the following cost estimates for the production phase REVS runs, by day:

<u>DAY</u>	<u>RUNS/DAY</u>	<u>CUs/DAY</u>	<u>AUs/DAY</u>	<u>COST/DAY</u>
1	2	100	120	\$ 4.20
2	8	500	580	\$ 20.30
3	2	175	195	\$ 6.20
4	1	50	60	\$ 2.10
5	3	275	305	\$ 10.68
6	5	225	275	\$ 9.63
7	1	150	160	\$ 5.60
8	3	270	300	\$ 10.50
9	1	110	120	\$ 4.20
10	6	450	510	\$ 17.85
11	1	90	100	\$ 3.50
12	4	390	430	\$ 15.05
13	3	175	405	\$ 7.18
14	3	425	455	\$ 15.93

TOTAL ESTIMATED COST = \$135.93

6.2.4.2.4 PERCAM Job Cost Estimates

The two PERCAM utilization studies contained in Paragraph 4.2 would entail estimated costs at LBL for comparable PERCAM operations as shown in Tables 6.4 and 6.5.

6.2.4.2.5 AUTOIDEF Job Cost Estimates

There are no sufficient utilization studies available for purposes of estimating the costs involved in AUTOIDEF operations at CDC ARPANET host configured along the lines of LBL.

6.3 NON-CDC ARPANET HOSTS PARTIALLY EVALUATED BY TRW

Preliminary studies indicate that several classes of ARPANET host machines are adequate to the hardware requirements for combined CPU speed and I/O rate imposed by the STE. These include the IBM 370/158 and higher series and the Univac 1100/40 and higher series. We also consider DEC VAX 11/780 ARPANET hosts in this section, but this machine is considered under Task 3 for stand-alone STE operations as well.

6.3.1 Hardware/Software Functional Requirements

The hosts shown in Table 6.6 are presented in terms of the degree to which they satisfy the further software/hardware requirements.

UCLA-CCN and ANL are both large IBM multi-processor hosts with more than adequate hardware/software configurations for STE operations. ANL supports

Table 6.4 PERCA Study Number One Cost Estimates (LBL)

- 20 ATTACKERS VERSUS ONE DEFENDER
- TWO MAN-WEEK EFFORT
 - ONE WEEK FOR THE LEARNING CURVE AND EVENT LOGIC TREES
 - ONE WEEK FOR COMPUTER RUNS AND DOCUMENTATION

COST ESTIMATES

CHECKOUT PHASE	RUN NO.	CPU SEC.	COST
	1	11.445	\$ 2.00
	2	20.161	\$ 3.53
	3	20.133	\$ 3.52
	4	20.720	\$ 3.63
	5	20.019	\$ 3.85
	6	21.199	\$ 3.71
	7	22.582	\$ 3.95
	SUBTOTALS	138.259	\$24.20

STUDY PHASE			
	8	22.210	\$ 3.89
	9	22.080	\$ 3.86
	10	22.005	\$ 3.85
	11	22.023	\$ 3.85
	12	21.722	\$ 3.80
	13	21.687	\$ 3.80
	SUBTOTALS	131.259	\$22.97

TOTAL 7600 SECONDS = 270

TOTAL COST STUDY ONE = \$47.00

Table 6.5 PERCAM Study Number Two Cost Estimates (LBL)

- 18 ATTACKERS VERSUS 20 DEFENDERS
- FOUR MAN-WEEK EFFORT
 - TWO WEEKS FOR MODULE BUILDING
 - ONE WEEK FOR MODEL VERIFICATION
 - ONE WEEK FOR PARAMETRIC AND SENSITIVITY ANALYSIS

COST ESTIMATES

MODEL BUILDING	RUN NO.	CPU SEC.	COST.
	1	8.492	\$ 1.49
	2	10.318	\$ 1.81
	3	16.191	\$ 2.83
	4	16.382	\$ 2.87
	5	4.599	\$.80
	6	18.918	\$ 3.31
	7	20.567	\$ 3.60
	8	20.381	\$ 3.57
	9	16.701	\$ 2.92
	10	38.583	\$ 6.75
	SUBTOTALS	170.072	\$29.76

VERIFICATION

11	22.219	\$ 3.89
12	20.719	\$ 3.63
13	20.917	\$ 3.66
14	11.911	\$ 2.08
15	12.675	\$ 2.22
16	22.435	\$ 3.93
SUBTOTALS	110.876	\$19.40

P&S ANALYSIS

17	50.271	\$ 8.80
18	41.235	\$ 7.22
19	41.404	\$ 7.25
20	42.746	\$ 7.48
SUBTOTALS	175.646	\$30.74

TOTAL 7600 SECONDS = 457

TOTAL COST STUDY TWO = \$80.00

Table 6.6 Non-CDC ARPANET Host Evaluation

	GOOD	MARGINAL	UNACCEPTABLE	UNDETERMINED
PERCAM	UCLA-CCN, CCA NOSC-CC, NWC, ANL, NUSC		LL	DTI
AUTOIDEF	UCLA-CCN, CCA, NOSC-CC, NWC, ANL, NUSC		LL	DTI
RSL/REVS	UCLA-CCN, CCA, ANL	NOSC-CC, NWC, NUSC	LL	DTI

only 1200 baud communications with ARPANET users but is willing to upgrade this aspect of their communications configuration to the requirements of a cost effective long term (=5 year) ARPANET software project. Both hosts can offer the new IBM Pascal compiler in their system libraries for purposes of REVS development. LL uses an Amdahl 470/V7 processor running under VM/370, an IBM type system, but does not support software development by non-local ARPANET users.

CCA, DTI and NUSC use DEC VAX 11/780 processors but, whereas NUSC runs the DEC VMS operating system, CCA and DTI run Paging UNIX (U. C. Berkeley, v.32). VMS supports DEC Pascal, an implementation already used in one REVS installation. Paging UNIX, however, does not as yet support an adequate production quality Pascal compiler. The NBS Pascal compiler, originally developed in a UNIX environment, is expected to be re-embedded into UNIX environments in the near future. However, the NBS compiler does not support GO TO constructs used by REVS for error recovery. Because of the popularity of the DEC VAX, the UNIX operating system, and the Pascal language, it is not unreasonable to expect that an adequate Pascal compiler will emerge in the near future. The availability of such a compiler is presupposed in our rating CCA a good STE candidate. DTI's status is uncertain because, as yet, it has no interface to the ARPANET that supports the necessary file transfer operations. NUSC has been rated marginally adequate because its VAX has a minimal main memory size (1 megabyte).

There is a widespread belief that Paging UNIX does not compare favorably with VMS as a VAX-hosted virtual memory operating system. In fact, this was the case with the initial versions of Paging UNIX which were more concerned to exploit the transportability of that system than to tune its interfaces to the VAX architecture. The paging performance of the latest version of Paging UNIX is within 10% of VMS's. It must be pointed out, however, that an unknown

degree of modification may be necessary to adapt REVS to interface with UNIX. Adaptation to VMS has already been done.

NOSC-CC and NWC both use Univac processors. Both have sufficient main and secondary memory for STE operations and provide Mike Ball's NOSC Pascal compiler in their user libraries. This compiler appears to satisfy the REVS compiler requirements detailed in Paragraph 2.2.6.1, except that DISPOSE is not supported. NOSC runs the Univac 1100 operating system while NWC runs the EXEC 8 operating system. Both operating systems have the functionality and utilities required for STE operations. However, the sites are listed as marginal for REVS support because of the need to modify the NOSC compiler. A compiler developed by the University of Copenhagen (Denmark) may be an acceptable alternative, but is not currently supported at any site.

6.3.2 Non-CDC Host Processor Slowdown Ratios

The desired slowdown ratio for STE hosts should be no worse than fifty-five to one relative to the canonical CDC 7600 installation at the BMDATC ARC. TRW has produced benchmark studies which compare the makes/models of the non-CDC hosts presented above. The performance comparisons are made in terms of a weighted measure of CPU speed and I/O rate. The following is a summary of the results of these studies and conversations with TRW systems personnel well-versed in the characteristics of the host machines:

<u>HOST MACHINE</u>	<u>HOST:CDC 7600 SLOWDOWN RATIO</u>
IBM 3033	<3:1
Univac 1100/82	<6:1
Univac 1110/40	<9:1
DEC VAX 11/780	7:1 on MIPS alone, I/O is known to increase this greatly.

This information, while informative for bounding the slowdown ratios, takes into consideration neither STE-specific operations nor host processor efficiencies available to them.

6.3.3 Host Utilization Considerations

The following particulars have been gathered from inputs provided by non-CDC host user service personnel. The general points about host utilization variations with time of year, day and week apply to non-CDC ARPANET hosts as well.

- CCA
 - As much as 50% of the system capacity is available on the average day.
- ANL
 - The processors always have some excess capacity.
 - One or two more interactive users could be accommodated during the day.

- The batch system saturates "softly" in that it can accommodate small jobs (500 Kbyte + 10 CPU sec) jobs at any time.

6.3.4 Host Job Scheduling and Billing Considerations

To date, there is no model study of STE utilization for non-CDC ARPANET system host machines otherwise adequate for the requirements imposed by the STE. In these cases, given an account of the job billing algorithm, policies and procedures of a representative non-CDC host and the tolerances in performance slowdown of a given host relative to the canonical CDC 7600 installations of the STE, we can still bound the job costs associated with STE operations at the host.

ANL systems personnel have provided us with an account of user service rates in effect since the end of July, 1980. These rates will be increased by fifteen percent some time in November, 1980. The precise billing and scheduling algorithms were not provided although the service rates give a good measure of STE job costs based on the utilization results of Section 4.

In this section, we present the service rates of interest to STE operations at ANL, estimate the costs associated with the jobs recited in our discussion of STE job costs at LBL, and conclude with a summary of the job scheduling parameter breakpoints of importance to STE operations. Note that our approach assumes that STE performance at ANL will be comparable to STE performance at LBL. This is not as unrealistic as one might at first think because the CDC 7600 and the IBM 3033 processors are actually closer to one another in performance than our "worst case" slowdown ratio presented in the last section would suggest.

6.3.4.1 ANL Service Charges of Interest to STE Operations

<u>INTERACTIVE SERVICES</u>			
	<u>CHARGING UNIT</u>	<u>PRIME TIME</u>	<u>NIGHTS AND WEEKENDS</u>
<u>CMS (IBM 3033)</u>			
Session Time	Hour	\$ 1.20	\$ 0.54
CPU Time	Hour	240.00	108.00
Disk I/O	EXCP	0.0006	0.00027
Storage	Kilobyte	0.282	0.1269
Occupancy	Hour		

WYLBUR (IBM 3033 and 370/195)

Session Time	Hour	\$ 1.20	\$ 0.54
CPU Time	Hour	240.00	108.00
Disk I/O	EXCP	0.0006	0.00027

BATCH SERVICESOS/MVT and ASP (IBM 3033 and 370/195)

CPU Time*	Hour	\$120.00
Wait Time*	Hour	60.00
Core x	Kilobyte	0.141
CPU TIME*	Hour	
Per Job Surcharge		0.15
Library Tape	Tape	0.50
Setup**		
Personal Tape	Tape	0.75
Setup**		
Disk Setup**	Disk	1.50

Note that the term EXCP refers to an I/O block (approximately 1680 to 3120 bytes) transfer. EXCPs involve a worst case "wait time" of 35 milliseconds.

Batch Priority Multipliers

Items marked * above have the number of charge units multiplied by the priority factor for the priority chosen for the job; those marked by ** are multiplied for Top priority only. The priority multipliers are:

Top	3.0
High	1.5
Normal	1.0
Low	0.9
Standby	0.8
Zero	0.8

GRAPHICAL OUTPUT

<u>PAPER</u>	<u>JOB</u>	<u>HOURL</u>	<u>FOOT</u>
Calcomp 580	\$2.00	\$4.50	\$.04
Calcomp 780	2.50	4.50	.04
Calcomp 936	4.50	7.50	.14
Versatec	None	None	.14

DIRECT ACCESS STORAGE

	<u>CHARGING UNIT</u>	<u>RATE</u>
Permanent Disks (PERM)	Track Day	\$.00354
Temporary Disks (TEMP)	Track Day	.00191
Timesharing Permanent Disks (TSPERM)	Track Day	.01158
Timesharing Temporary Disks (STEMP)	Track Day	.00354
Permanent 2314 Disks (LONG2314)	Track Day	.00648
Database Disks (DATABASE)	Track Day	.00458
CMS Virtual Disks (MINIDISK)	Megabyte Day	.24000
Timesharing Database Disks (TSDATA)	Track Day	.01610
Migration Disks	Track Day	.00229
Data Cells	Track Day	.00048
Storage of Setup Disk	Disk Month	13.00
CMS Minidisk Restore	CMS Minidisk	1.30

MAGNETIC TAPE SERVICES

Tape Storage	Tape Month	\$.65
Tape Save Request Processing	Request	2.50
Withdrawal of New Tape from AMD Stock	Tape	25.00
Withdrawal of Tape from AMD Library	Tape	25.00
Use of Tape to Send Expired Datasets to User	Tape	25.00

<u>ARPANET USAGE CHARGE</u>	Hour	\$10.00
-----------------------------	------	---------

6.3.4.2 Estimating STE Costs at ANL

As in the case of STE job cost estimates at LBL, we consider REVS and PERCAM costs. For REVS, we consider file storage charges as well as installation and production phase costs as determined by the utilization results of the Interim Report.

6.3.4.2.1 REVS File Storage Costs at ANL

Assuming that the installation phase requires a week and that approximately 3.35 megabytes of mass storage file space is required, we arrive at the following charge given the ANL rate of \$.24 per megabyte per day:

$$\begin{aligned} &\text{REVS INSTALLATION MASS STORAGE CHARGE} \\ &= 3.35 \times 7 \times .24 \\ &= \$5.60 \end{aligned}$$

The production phase mass storage files require approximately 12 megabytes of disk space, which yields a charge of:

REVS PRODUCTION MASS STORAGE CHARGE

= 12 x .24
 = \$2.90 per day
 = \$87.00 per month

The databases (up to 16 of them at an STE site) used in REVS applications each require between 1.5 and 4.5 megabytes of mass storage, which yields:

REVS APPLICATION DATABASE CHARGE

=> 1.5 x .24 = \$.36 per project per day
 <= 4.5 x .24 = \$1.08 per project per day

6.3.4.2.2 REVS Installation Charges at ANL

Given the service rates presented in the previous section, we can estimate the cost of installing REVS at ANL as follows, assuming a better than worst case, 1:1 slowdown ratio for the ANL IBM processors relative to the CDC 7600:

<u>SERVICE</u>	<u>UNITS</u>	<u># UNITS</u>	<u>RATE</u>	<u>CHARGE</u>
CPU Time	Hour	.1667	\$120.00	\$ 20.00
Wait Time	Hour	.5	\$ 60.00	\$ 30.00
Core x	Kilobyte	170.67	\$ 0.141	\$ 24.06
CPU Time	Hour			
Core x	Kilobyte	512	\$ 0.141	\$ 72.19
Wait Time	Hour			
Per Job Surcharge	Job	52	\$ 0.15	\$ 7.80
TOTAL =				\$154.00

Here we have assumed job resource requests of normal priority, one megabyte of main memory, and a 35 millisecond per EXCP of memory wait time. The key utilization data, you will recall, were 604 CPU seconds, 14.54 megawords (60-bit word) of I/O, and 52 separate jobs spanning a period of 5 work days. Note that this figure is comparable to our result for the STE installation charge at LBL.

6.3.4.2.3 REVS Production Phase Costs at ANL

As our results suggest, we assume that a fifth of the total charge for a given STE job is due to the CPU time charge. This yields the following breakdown of job costs for the REVS production phase jobs presented in Paragraph 4.1.

<u>DAY</u>	<u>RUNS/DAY</u>	<u>CPU SECS.</u>	<u>COST/DAY</u>
1	2	20	\$ 3.63
2	8	100	17.87
3	2	35	6.13
4	1	10	1.82
5	3	55	9.62
6	5	45	8.25
7	1	30	5.15
8	3	54	9.45
9	1	22	3.82
10	6	90	15.90
11	1	18	3.15
12	4	78	13.60
13	3	35	4.62
14	3	85	14.62
TOTAL			<u>\$120.00</u>

This estimate is also comparable to the REVS production phase/cost estimates at LBL.

6.3.4.2.4 PERCAM Job Cost Estimates at ANL

The two PERCAM utilization studies contained in Paragraph 4.2 present the results of PERCAM job runs in CDC 7600 CPU seconds. Assuming a 1:1 slowdown ratio for the ANL IBM processors relative to the CDC 7600 and 5:1 total:CPU-time charge ratio, we have the cost estimates for PERCAM runs at ANL shown in Tables 6.7 and 6.8.

6.3.5 Job Scheduling Parameter Considerations at ANL

We have already mentioned that "small" batch jobs at ANL are those which request less than 10 seconds of CPU time and less than 500 kilobytes of main storage. Small jobs will be run during prime time hours although they will encounter some delay. Some important breakpoint values for CPU time and main memory are as follows:

<u>QUEUE</u>	<u>CPU TIME</u>	<u>MAIN MEMORY</u>	<u>PRIORITY</u>
Express	<= 2 min.	<= 250 Kbytes.	Top
.. variable ..		<= 650 Kbytes.	High
		<= 1500 Kbytes.	Normal
		<= 2 Mbytes.	Low
Standby	<= 15 min.	<= 3 Mbytes	Standby
	> 15 min.	> 3 Mbytes.	Zero

Table 6.7 PERCAM Study Number One Cost Estimates (ANL)

- 20 ATTACKERS VERSUS ONE DEFENDER
- TWO MAN-WEEK EFFORT
 - ONE WEEK FOR LEARNING CURVE, EVENT LOGIC TREES
 - ONE WEEK FOR COMPUTER RUNS AND DOCUMENTATION

COST ESTIMATES

CHECKOUT PHASE	RUN NO.	CPU SEC.	COST
	1	11.445	\$ 1.91
	2	20.161	\$ 3.36
	3	20.133	\$ 3.36
	4	20.720	\$ 3.45
	5	22.019	\$ 3.67
	6	21.199	\$ 3.53
	7	22.582	\$ 3.76
	SUBTOTALS	138.259	\$23.04
STUDY PHASE	8	22.210	\$ 3.70
	9	22.080	\$ 3.68
	10	22.005	\$ 3.67
	11	22.023	\$ 3.67
	12	21.722	\$ 3.62
	13	21.687	\$ 3.61
	SUBTOTALS	131.259	\$21.90

TOTAL ANL CPU SECS = 270

TOTAL COST STUDY ONE = \$45.00

Table 6.8 PERCAM Study Number Two Cost Estimates (ANL)

- 18 ATTACKERS VERSUS 20 DEFENDERS
- FOUR MAN-WEEK EFFORT
 - TWO WEEKS FOR MODULE BUILDING
 - ONE WEEK FOR MODEL VERIFICATION
 - ONE WEEK FOR PARAMETRIC AND SENSITIVITY ANALYSIS

COST ESTIMATES

MODEL BUILDING	RUN NO.	CPU SEC.	COST
	1	8.492	\$ 1.42
	2	10.318	\$ 1.72
	3	16.191	\$ 2.70
	4	16.382	\$ 2.73
	5	4.599	\$.77
	6	18.918	\$ 3.15
	7	20.567	\$ 3.43
	8	20.381	\$ 3.40
	9	16.701	\$ 2.78
	10	38.583	\$ 6.43
	SUBTOTALS	170.072	\$28.50
VERIFICATION	11	22.219	\$ 3.70
	12	20.719	\$ 3.45
	13	20.917	\$ 3.49
	14	11.911	\$ 1.99
	15	12.675	\$ 2.11
	16	22.435	\$ 3.74
	SUBTOTALS	110.876	\$18.48
P&S ANALYSIS	17	50.271	8.38
	18	41.235	6.87
	19	41.404	6.90
	20	42.746	7.12
	SUBTOTALS	175.656	\$29.26

TOTAL ANL CPU SECS = 457

TOTAL COST STUDY TWO = \$76.00

6.4 HONEYWELL 6180 HOSTS

The only Honeywell 6180 ARPANET hosts are RADC-MULTICS and MIT-MULTICS. Both of these hosts are adequate to the hardware/software requirements of PERCAM and AUTOIDEF. However, neither host supports a production quality Pascal compiler at present. Honeywell plans to release and support a compiler for a language called PYXIS which closely resembles Pascal but which does not meet the requirements imposed by REVS on a candidate Pascal compiler. Apart from software releases supported by Honeywell, there is no certain source of an adequate Pascal compiler for RADC-MULTICS and MIT-MULTICS. We therefore do not regard either host as adequate to STE operations.

6.5 REVS RUN COSTS AT NADC

Table 6.9 presents run cost and total central processor (CP) time for selected REVS test cases run on the NADC "B" machine during REVS installation at NADC in May 1979. Comparable execution times on the CDC 7600 at the U. S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC) Advanced Research Center (ARC) are also shown, where available. The NADC configuration executes 5 to 6 times slower than the 7600. The NADC costs include line printer and service charges.

An approximate cost estimating relationship derived from the data of Table 6.2 is:

$$\text{COST} = \$6.50 + \$0.42 \text{ per CP/sec}$$

Considering an average 5.7:1 slowdown ratio between the NADC machine and a CDC 7600, we multiplied the CPU times used in the LBL and ANL estimates for REVS production runs by 5.7 and applied the above cost rule. The result was a \$1900 estimate for NADC versus \$136 and \$120 for LBL and ANL respectively. Review of the LBL and ANL estimates showed no errors that could explain the 15:1 cost difference between NADC and the other sites. However, the NADC charges represent total cost, including all peripheral services, and presume that service costs are roughly proportional to CP time.

The only conclusive way to verify the relative costs would be to make actual REVS runs, or benchmark runs with a similar program (in terms of resource usage profile) at LBL, ANL, and NADC. In the absence of such runs, we can only consider the NADC data to be a "high" cost estimate, and the LBL/ANL data to be a "low" cost estimate.

Table 6.9 NADC REVS Costs

TEST CASE	NADC \$ COST	NADC CP-SEC	ARC 7600 CPU-SEC
1. RADX TEST DB-ANALYZE DATA FLOW	\$43.03	90.69	--
2. SAMPLE SIMULATOR (RUN 1)	37.54	88.78	--
3. SAMPLE SIMULATOR (RUN 2)	37.60	33.86	--
4. BUILD NUCLEUS	16.42	9.20	4.55
5. BUILD TRACKLOOP	57.17	115.52	20.18
6. BUILD RADX TEST DB	25.44	45.85	9.43
7. SAMPLE SIMULATOR (RUN 3)	34.53	86.10	--
8. LIST ALL-BALLOON TRACK	11.74	16.05	3.20
9. AA TEST	15.42	16.51	3.93
10. LIST PERMISSIONS (RADX 3)	5.98	5.15	1.31
11. LIST RSL (RADX 2)	23.48	40.57	7.84
12. TRACKLOOP ERROR DETECT (RADX 6)	32.11	53.42	10.81
13. TRACKLOOP ANALYZE ALL	22.56	44.34	7.44
14. TRACKLOOP ANALYZE DATA FLOW	58.09	127.76	21.28
15. TRACKLOOP PUNCH & PLOT (RADX 4)	30.39	55.70	9.40
16. BUILD BALLOON TRACK	24.47	44.14	--
17. LIST ALL-TRACKLOOP	23.40	37.17	7.63

6.6 ARPANET HOST CONCLUSIONS

As discussed in the previous paragraphs, there are three CDC ARPANET nodes and five non-CDC ARPANET nodes that have sufficient resources to support the STE, and which either have adequate Pascal compilers or could potentially be provided with adequate compilers to support REVS.

One of the CDC nodes, NADC, already has two STE tools, REVS and PERCAM, installed. The addition of AUTOIDEF would not appear to be difficult, provided that the tool is portable, as claimed. The costs of running at NADC may be higher than similar costs at other sites, but this cannot be verified with available information. The other two CDC nodes, BNL and LBL, operate with versions of the SCOPE operating system, under which REVS was developed. However, since these nodes use different versions of SCOPE, and may have hidden site dependent features, installation of REVS at either site may require one-half to two man-years of adaptation effort, depending on the details of the operating system. Optimistically, the adaptations would be minimal. Adaptation to run at nodes using NOS or NOS/BE would definitely require effort at the upper end of the range.

Of the non-CDC nodes, ANL and UCLA-CCN have the most powerful capabilities and, apparently, adequate Pascal compilers. Previously, REVS has not been transported to an IBM mainframe, but building on the experience of previous CDC transfers and transport to the VAX 11/780, the job could be achieved with less than two man-years of effort.

A similar transfer could be made to the Univac nodes, NOSC-CC and NWC, but there are remaining issues of Univac Pascal compiler adequacy. If existing compilers have to be significantly modified to support REVS, the transfer cost would increase substantially.

DEC VAX 11/780 hosts are just beginning to appear on the ARPANET, and several are contemplated. Many of these will run under the Paging UNIX operating system, and would require an amount of STE software modification that cannot be estimated without a detailed analysis of UNIX beyond the resources of this study. The one existing node operating under the VMS operating system, NUSC, is marginal because of its 1 megabyte memory. Upgrade of the memory size to 2 megabytes or more would make NUSC an attractive STE site, because REVS is being transferred to the VAX 11/780 by BMDATC, thus saving adaptation costs for the STE. It is highly probable that many of the popular 11/780s will be connected eventually to the ARPANET, and REVS can be readily installed on those nodes with VMS and sufficient memory. Installation of AUTOIDEF does not seem to be difficult, and PERCAM is operating on the VAX at BMDATC.

The benefits of using an existing ARPANET site must be weighed against the potential disadvantages. The disadvantages are: 1) potential lack of host availability when needed; 2) inability to tailor the host specifically to STE needs; and 3) inability to make classified runs without establishing a secure subnet and acquiring Private Line Interfaces (PLIs) at \$55,000 + per copy.

7.0 SURVEY OF ALTERNATE STE HOSTS

This section presents the results of Task 3 of the STE Study. The main objective of Task 3 was to survey commercially available, off-the-shelf computer systems, using a mainframe of a type that is being or has been interfaced to the ARPANET, with the capability of supporting the STE. A second objective was to investigate the security implications of operating the STE in a single-level, dedicated security environment.

Because the dominant families of large mainframes were effectively considered under Task 2, in Task 3 we concentrated on high-performance, economical minicomputers, and midcomputers. After initial analysis we concluded that current 16-bit minicomputers could not access the large address space required by REVS without unacceptable performance penalties. Therefore, more detailed analysis concentrated on 32-bit midcomputer families. The only appropriate machines of this type currently on the ARPANET are DEC systems. For comparative purposes we broadened the investigation to consider other major midcomputer systems that might be interfaced to ARPANET in the future.

Paragraph 7.1 discusses methods of investigation for Task 3. Paragraph 7.2 summarizes features of the systems investigated and discusses issues involved in the analysis. Paragraph 7.3 presents our rationale for selection of the DEC VAX 11/780 as the preferred system. Paragraph 7.4 discusses color graphics capability for the STE. Security issues are summarized in Section 8.

7.1 METHODS OF INVESTIGATION: TASK 3

Information on the functional characteristics of a wide range of 32-bit midcomputers was gathered from TRW's Minicomputer and Information Technology Laboratory which maintains current data on available minicomputer hardware and software. The criteria used in selecting alternate midcomputer host architectures were the same as the hardware and software functional requirements imposed upon candidate ARPANET system hosts under Task 2 except for additional consideration of stand-alone and potentially secure STE operations as discussed in Section 8.0. We also cite trends and particulars that were provided by vendor personnel.

Among midcomputers, we selected for initial consideration only those which satisfy the following criteria:

- Availability of operating systems which meet the functional requirements for an STE support environment.
- Network Communications hardware/software support within such an operating environment for potential ARPANET operations.
- Availability of FORTRAN and Pascal compilers within such an operating environment which satisfy the translation requirements of each tool of the STE.

These criteria significantly reduced the number of candidate host machines for stand-alone STE operations. Cost considerations and the availability of potentially adequate Pascal compilers limited the class of candidates to the following machines:

Digital Equipment: VAX 11/780
Harris: H800
Perkin Elmer: 3244, (Interdata) 8/32
Prime: 750
SEL: 32/77

Other midcomputers are known to either lack Pascal compiler support or to exceed the cost of these machines by a large factor.

7.2. COMPARATIVE DATA ON ALTERNATE STE HOST MACHINES

Table 7.1 presents the features that qualify each of the candidate STE midcomputer host machines:

Table 7.1 Candidate Midcomputer Hosts

	MIPS	I/O RATE MB/SEC	PRICE	W/MBs	NTWK COMM
DEC VAX 11/780	1.5	8 (massbus); 1.5 (unibus: 13.3 (dma channel)	\$225,000	2	DECNET (OPTIONAL)
Harris H800	2.6	1 (slow device); 19 in, 7.9 out (high speed dev)	\$183,000	1	
PE 3244	1.4	4 DMA Channels 10 - 40 MB/SEC	\$169,000	2	DATUEB CMS CORP.
PE 8/32	2+	1 DMA Channel 8.3 MB/SEC	\$120,000	1	DATUEB CMS CORP.
Prime 750	.895	High Speed Device 8 input, 5 output	\$186,000	2	PRIMENET (OPTIONAL)
SEL 32/77	.98	High Speed Device 3.2: 1 DMA Channel @ 26.6 M/sec	\$110,000	2.3	IN HOUSE X.25+

The table does not reflect certain important facts about the candidate midicomputer STE host systems. The available Pascal compiler support for some of them is neither as robust nor as field proven as it is for others. Moreover, the network support software provided by any two of the vendors differs in point of function and design objectives. The main concerns are that global as well as local networking be supported and that the global networking support be general enough at critical module levels to accommodate extensions to incorporate ARPANET protocols if these are not already supported. A further concern is the extent to which a choice of a given midicomputer system furthers RADC's objective of STE technology transfer. Here the sheer number of such midicomputer systems available to potential STE users is a primary consideration.

From this standpoint, the following caveats should be registered; first, those regarding Pascal support. The compilers available to the candidate midicomputer systems differ in their age and exposure to production environments. DEC Pascal has been available for less than a year. Harris Pascal has not been released yet and is in the latter phases of implementation. Prime and SEL Pascal are also moderately new. By comparison, the candidate Pascal compiler for the Perkin Elmer machines has been around for some time, originally in a Univac incarnation. Documentation of the various compilers typically touches on the more salient language features and operating system interfaces that are supported but provides little in the way of critical information such as the degree of optimization achieved in the generated target midicomputer machine code or the friendliness of the host operating system to the Pascal software development process. For instance, the SEL Pascal user's manual focuses primarily on the grammar which the SEL compiler recognizes and mentions very little about the Pascal user's interface to the operating environment, benchmark comparisons of SEL Pascal to SEL compilers for other languages, limitations on the amount of machine code used to realize primitive types, procedures, compiler and run-time stack space, and other "messy details" essential to its usefulness in a production environment. Much the same is true of DEC Pascal; as experiences in porting RSL/REVS to the VAX have shown. Moreover, new compilers, like all other significant pieces of new software are bound to have bugs. These concerns put the NOSC Pascal compiler for the Perkin Elmer machines in a good light. However, the Perkin Elmer compiler, like its NOSC ancestor, does not support the DISPOSE service.

Next, there are caveats about networking software. DECNET was originally designed and implemented for local networking within a tightly coupled cluster of communicating processors. Unlike PRIMENET, DATUEB and the in-house development of networking software under way at Harris, which also support local networking, DECNET does not support global networking among distant host processors. Moreover, the global support provided by the others conforms to the X.25 packet-switching standard. Apart from this advantage, there is some question about the generality of the global networking support provided by the non-DEC host systems. PRIMENET, for instance, may well be highly modularized and parameterized since it supports

a wide variety of packet-switching network standards, viz., the United States' TELENET and TYMNET, the Canadian DATAPAC, Great Britain's International Packet Switching Service (IPSS), and France's TRANSPAC as well as the European Packet Switching Network, EURONET. Presumably, ARPANET support would not be too difficult to achieve within the Prime networking software environment. By comparison, the documentation on networking software for Perkin Elmer, SEL, and Harris systems does not indicate the sort of generality which Prime has obviously advertised. But this may merely reflect a lack of appropriate data which, in turn, may reflect the fact the PRIMENET has been on the market longer than the networking software provided by the other vendors that support global networking.

Finally, there are considerations due to the objective of STE technology transfer. This constraint may well be the most significant of all. There are more DEC VAX 11/780s than there are of any of the other candidate midcomputer STE host systems. Moreover, although DEC neither developed nor supports it, there exists a substantial and increasing amount of global networking software support of the ARPANET protocols. This software has been developed largely by ARPANET host systems personnel operating under the standard DEC VMS operating system or the Berkeley Paging UNIX operating system. Most of these personnel work in non-secret environments and might thus be in a position to transfer their networking software technology in the interests of STE technology transfer.

7.3 MIDICOMPUTER HOST SELECTION

On consideration of the foregoing issues, we believe that the DEC VAX 11/780 is currently the best candidate to host the STE. Its immense popularity makes it an appropriate vehicle for widespread technology transfer affordable by many users, and seems to ensure availability of better ARPANET support. Because REVS and PERCAM are hosted on the VAX, software transfer costs are confined to site installation changes and transfer of AUTOIDEF. Early development of ADA compilers for the VAX will permit an early language conversion for STE tools, if desired. On the whole, although I/O bottlenecking is known to exist, the VAX has high performance and sufficient memory capacity, plus an adequate Pascal compiler.

The next best choices are the Perkin Elmer 3244 and Prime 750 systems. The PE 3244 offers higher performance at a lower price, and is a class of machine more widely used in the DoD community. However, the existing Pascal compiler needs augmentation since it lacks the DISPOSE function. Kansas State University has recently developed a Pascal compiler for Perkin Elmer machines, but details of the language implementation has not been confirmed. The Prime system is slower and costlier than the Perkin Elmer system but has several attractive features. First, it is aimed at a time-sharing, commercial data base environment. Its native data base management system uses a CODASYL schema, as does the REVS and AUTOIDEF tools. Second, it appears to have an excellent Pascal compiler, meeting REVS requirements. Third, it supports a wide variety

of commercial networking standards and may be easily adaptable to an ARPANET interface. However, Prime aims mainly at the commercial market, and the DoD community seems to have overlooked Prime systems to date.

We rank the Harris and SEL systems behind the above candidates, and prefer the Harris over the SEL because of its exceptional CPU speed and I/O rate combination. However, neither of these systems have been connected to the ARPANET, they are not widely used as yet, and there are many questions remaining about their Pascal compilers and networking capabilities.

A DEC VAX 11/780 configuration should include at least two megabytes of main memory. Basic DEC VAX systems include a single disk and a single tape unit of the purchaser's choice, two Massbus adapters and a DZ11A 8-line asynchronous multiplexer. The low speed TEE-16 (45 in/sec) tape unit seems adequate for the STE. Disk configurations should be chosen for overall system flexibility and a view toward long-term needs. Three RM-03 disks are a minimum requirement, but larger RM-05 disks would offer growth potential. This disk configuration is a matter of preference as long as the disks are removable (for secure operations, a requirement). Among DEC users the disk drives manufactured by CDC (e.g., the RM series) seem to enjoy a better reputation than others.

The LP11-DA line printer (132 column, 96 character, 660 lines/min.) is adequate for STE use. Either VT100 or VT52 alphanumeric terminals are acceptable. If a card reader is desired, the CR11 (300 cards/min.) seems adequate. Software should include the VMS operating system, the FORTRAN compiler, and the Pascal compiler.

7.4 STE COLOR GRAPHICS TERMINALS

Candidate color graphics terminals meeting the STE requirements in Paragraph 3.1.4 were selected from the Data Pro EDP Buyer's Bible and by contact with various vendors. Four terminals that were in the desired cost range were first identified:

- Tektronix 4027
- DEC VS11
- Ramtek 6200A
- Three Rivers CVD/2.

The Tektronix 4027 is the preferred choice. It meets all of the requirements in Paragraph 3.1.4 and seems to be a popular color terminal at ARPANET nodes (see Appendix A). It is the only "low cost" color terminal with 480 x 640 resolution, compatible with that at the BMDATC ARC. This format also seems ideal for the AUTOIDEF displays, and conversion from a Tektronix 4014 to a 4027 would probably be easier than to a terminal from another manufacturer. The 4027 can be operated both locally and remote.

The DEC VS11 is just becoming available. Its advantages are design for use with DEC computer systems and capability to use a 19 inch color monitor. The principal disadvantage is its 512 x 512 resolution. REVS and AUTOIDEF displays would have to be reformatted to fit a square format. A Ramtek 6200A has lower resolution (256 x 512) than is desirable, but TRW experience with it on another contract indicates that this is on the borderline of adequacy for STE use. No detailed information was available on the CVD/2. However, its cost for comparable capability, lack of cursor display, and uncertain interface to a host probably make it a fourth choice.

A Tektronix 4027 STE graphics installation should have two 4027 terminals sharing a Tektronix 4632 hard copy unit (with 06 enhanced gray scale option). The 4027 optional features suited to STE use are:

- Option 21 - 16K bytes additional display memory.
- Option 28 - 144K bytes additional graphics memory.
- Option 31 - Character Set Expansion.
- Option 32 - Ruling Characters.
- Option 42 - Video Hardcopy/Video Output Interface.

The approximate cost per 4027 in this configuration is \$16K, and the 4632 hard copy unit cost is approximately \$5K. Thus, total cost for the STE graphics installation is about \$37K, plus cost of modems and cables dependent on the location of the terminals, and the interface with the STE host computer. Monthly maintenance costs are extra. (Note: prices are subject to change and, in 1980, are unstable for all types of DP equipment.)

8.0 STE SECURITY CONSIDERATIONS

This section briefly examines some of the implications of operating the STE in a single-level, dedicated mode (i.e., where a single classification level is concerned and all authorized users have need-to-know and clearance for all data in the system). This mode is a limited subset of the general multi-level security problem. TRW has prepared a detailed treatment of requirements for multi-level computer security systems under a separate contract. The reader is encouraged to consult [10] for appropriate requirements for single-level dedicated mode systems and pertinent DoD regulations.

8.1 GENERAL SECURITY CONSIDERATIONS

Security in current classified data processing systems is maintained through use of conventional security measures -- clearance of users to the highest level, physical security of the computer installation and terminal locations, administrative security procedures governing access to and use of the data processing systems, encryption of communications, emanation security -- plus processing limitations and access control software. With these security measures, processing of compartmented information is limited by security regulations - DoD 5200.28 and DCID 1/16 -- to one or the other of two security modes -- dedicated, where all users with access to the system have both a security clearance and need-to-know for all classified material then contained in the system, and system high, where all users with access to the system have a security clearance for the highest classification and most restrictive type of material contained in the system but at least some users do not have a need-to-know for all classified material contained in the system. In either of these modes, access to the system by users not cleared for all compartments or classifications is not allowed. Because computer systems processing compartmented data must be either dedicated completely to one or the other of these modes or scheduled for separate processing periods for different compartments, timeliness of processing is limited and operating costs are increased.

Contemporary computer systems do not contain the security controls needed to allow processing in a true multi-level security mode and they generally contain security vulnerabilities allowing evasion of any security controls. The principal vulnerabilities are in the operating system software, which must control concurrent processing and resource sharing for a large number of users, provide user services, and interface directly with the hardware. The complexity of operating systems is such that their structure and functioning has not been well understood, leading to design and implementation errors and hidden capabilities not specified in the design. A further complication is that operating systems are tested by the manufacturer on a limited number of machine configurations, but most users have a configuration differing in some way from the one on which the operating system was tested.

Most contemporary resource sharing computer systems are not secure because security was not a requirement of the initial hardware and software design and because there was not a generally accepted computer security definition usable as a basis for a secure design. Since the Air Force has effective means for implementing personnel, physical, communication, emanation, and procedural security, the heart of the computer system security problem is the security certification of the access controls in the operating system and supporting software.

8.2 STE IMPLICATIONS

An STE site with secure processing requirements will nominally alternate between secure and open access processing modes. When the STE is operating in a single-level dedicated mode, compartments within that mode are not an issue. Therefore, the problem is that of precluding access, direct or indirect, to any and all parts of the system by unauthorized user while the STE is performing operations on classified data.

The most obvious avenues of penetration can be closed by traditional physical and emanation security measures (e.g., limited personnel access, positive disconnect of all external communication lines, storage of classified data on separate dismountable media, and electromagnetic shielding of the facility where required). However, more indirect avenues of access, through the operating system, are more difficult to close with confidence.

In the absence of an expensive security kernel within the operating system, the only high-confidence means of preventing indirect access is to partition the system operating in classified mode and the system operating in open access mode into two disjoint systems. This would entail separate copies of the operating system for each mode, purging of all residual data at mode transitions, termination and recreation of the system at mode transitions under the appropriate OS copy, and physical removal of all media containing classified mode programs and data, including the OS copy, during open access mode, under strict operator procedures.

9.0 REVS IMPLEMENTATION LANGUAGE CONVERSION EVALUATION

The purpose of Task 4 of the STE study was to evaluate the feasibility of rewriting REVS, currently implemented in Pascal, in one of the DoD approved Higher Order Languages (HOL). The candidate languages are as follows:

- FORTRAN
- COBOL
- JOVIAL (J73) (MIL-STD-1589A)
- ADA.

A brief description of these languages are presented in Section 9.1.

In Section 9.2, we identify the current REVS features/capabilities which would be affected if REVS were to be converted to one of the candidate languages. A discussion of the ability of each candidate language to adequately support current REVS features/capabilities is also provided.

In Section 9.3, we discuss the cost (person-months) and schedule estimates for a feasible conversion. Also, we identify any software tools which could be utilized/developed to aid in accomplishing a cost-effective conversion.

In Section 9.4, we address the issue of the affect of an alternate implementation language on the life-cycle costs of REVS in the STE where the tools will be under continuous enhancement and modification. Finally, in Section 9.5, we summarize the evaluation and present our recommendation.

9.1 CANDIDATE LANGUAGE SUMMARY

This section contains a brief description of the design concepts and implementation techniques of the candidate languages.

FORTRAN Language

A FORTRAN program consists of a main program and a set of subprograms, each of which is compiled individually, with the object programs linked during loading. Each subprogram is compiled into a strictly statically allocated area containing the compiled executable code, system-defined data areas, and global data areas (COMMON blocks). No run-time storage management is provided. Subprograms communicate only by accessing COMMON blocks, passing parameters (call by reference), and passing control through non-recursive, non-nested subprogram calls. Data structures consist of multi-dimensional homogeneous (length and type of each element in the structure is the same), fixed-size arrays.

COBOL Language

COBOL is a highly structured language designed especially for business applications. An important characteristic is the English-like syntax which

makes most programs readable enough to be largely self-documenting. Numerous optional "noise words" are provided in the language to improve readability. The language design is based on static run-time structure (no dynamic run-time storage management). Subprograms are allowed but with these restrictions:

- Data can be shared only by passing parameters (call-by-name).
- Files cannot be shared between a main program and a subroutine (must be explicitly opened/closed by each).
- Subprograms cannot be nested or called recursively.

Most COBOL programs are written as a single routine using a common global referencing environment for the entire program. The PERFORM statement allows the COBOL programmer to organize code into functional units that can be invoked from different points within the same program unit. With this feature, true subprograms (with parameters and local variables) are infrequently used. Multidimensional homogeneous (length and type of each item of the structure is the same), and heterogeneous (structure items can have varying lengths and types) fixed-size data structures can be constructed.

JOVIAL J73 Language

J73 is a "block-structured" language composed of a main program, any number of COMPOOLS (common pool of data/subprogram declarations sharable between separately compiled subprograms), and a set of subprograms. Subprograms can be either disjoint, having no portion in common, or nested (static nesting), one subprogram completely enclosing the other. Subprograms can also be recursive. Dynamic run-time storage management is provided to support recursion since the maximum number of recursive activations, and hence the additional storage space, cannot be determined at compile time. Subprogram parameter passing is accomplished by call-by-value, call-by-reference, or call-by-value-result.

Like Pascal, J73 is a "strongly typed" language. Every data object declared in a program must be associated with a "type", which defines its logical properties and the operations that can be performed on the object. J73 supports complex multi-dimensional homogeneous/heterogeneous data structures.

ADA Language

The proposed ADA programming language is also a "block-structured" language being designed in accordance with DoD requirements. It is influenced by the Pascal and JOVIAL family of languages. An ADA program is a sequence of higher level program units, which can be compiled separately. Program units may be subprograms (which define executable algorithms), package modules (which define collections of entities), or task modules (which define concurrent computations). Subprograms can be nested (statically) and can have parameters (call-by-value, call-by-reference, and call-by-value-result). Also, subprograms can be recursive (dynamic nesting). A package module can be used to define a common pool of data (like JOVIAL COMPOOLS), or a collection of related subprograms.

ADA is also a "strongly typed" language requiring each data object to be associated with a specific "data type". Like JOVIAL J73, multi-dimensional homogeneous and heterogeneous collections of data items can be defined. Also, like Pascal, complex data structures can be created dynamically during program execution (dynamic run-time storage management).

9.2 CONVERSION IMPACT UPON EXISTING REVS FEATURES

The features and capabilities of REVS which are dependent (either directly or indirectly) upon the current implementation language, Pascal, are presented in this section, with an evaluation of the adequacy of each candidate language to support these features/capabilities.

9.2.1 REVS Program Architectures

REVS is a highly structured software program. The goal of the REVS designers was to separate complex operations into several well-defined functions (modules), each of which in itself could be declared to be correct by inspection. Each function, or module, is decomposed into smaller and smaller units (procedures) until the lowest level of the function is specified. Pascal, a "block-structured" language, provided the original designers with the module features desired. A block (or procedure) is allowed to have one or more procedures completely enclosed (statically nested) within itself.

The usefulness of statically nested procedures (i.e., subprograms) lies not just in the potential it provides for modularizing the computations, but in its ability to protect data that are the exclusive concern of a specific function/module from encroachment or contamination by other modules. For example, data declared in a procedure is said to be local to that procedure, and shareable (global) with all its' nested (or inner) procedures, but not shareable with outer or non-nested procedures. If the same data name is declared in both an outer and inner procedure, the outermost variable becomes inaccessible to the inner procedure.

The current implementation of the REVS program itself consists of 1108 Pascal procedures with nine levels of static nesting.

FORTRAN Implementation

A FORTRAN implementation would require that REVS' current modular architecture, utilizing statically nested subprograms, be redesigned since FORTRAN does not support this feature. Each nested subprogram would have to be unnested, with careful attention given to the shared data. For example, a Pascal data object declared in a procedure is shared by all procedures nested within that procedure. A FORTRAN implementation would require that all shareable data be made global (declared by COMMON statements), or passed as parameters between subprograms. FORTRAN global data is allocated a fixed amount of storage space at compile time. Since all data (approximately 5000 Pascal identifiers requiring 209K bytes of storage space) in the current implementation of REVS are allocated dynamically in reuseable space (see Section 9.2.2), a FORTRAN implementation would significantly increase REVS run-time memory requirements.

A FORTRAN implementation would require a total redesign of the REVS architecture plus increase the run-time storage space requirements, without adding additional capability or features.

COBOL Implementation

A COBOL implementation of REVS would also require a total redesign, and because of COBOL's English-like syntax which makes programs self-documenting, a significant amount of additional code would be produced, increasing conversion costs.

The COBOL language does not provide the nested subprograms, and it limits data sharing between subprograms to parameters passing only. Therefore, REVS would have to be re-written as a single COBOL program with all 5000 data identifiers declared global in the DATA DIVISION section (increasing run-time memory requirements), and all existing REVS procedures (1108 of them) replaced by in-line paragraphs of code executable by the COBOL PERFORM statement.

JOVIAL (J73) Implementation

The current REVS modular architecture could be implemented without change with JOVIAL J73, since statically nested subprograms are supported.

ADA Implementation

The current REVS modular architecture can be implemented without change with the proposed DoD standard language, ADA.

9.2.2 Dynamic Storage Management

The REVS implementation language provides for two types of memory storage management; static and dynamic. Static management is utilized during program compilation to allocate a fixed amount of space required for the program instructions and input/output buffers. The storage space required by the program's data structures and variables is not allocated until required during program execution (dynamic allocation). Dynamic storage management involves the reuse of storage space for multiple purposes during program execution. Two types of dynamic storage allocation are used in the current implementation of REVS -- stack and heap allocation.

At the start of REVS execution, a sequential block of memory is reserved for use as a dynamic run-time stack (120K bytes). Storage space is allocated on the top of this stack automatically (stack storage management) each time a procedure is executed/activated. This space (activation record) is used to record procedure-linkage information, the temporaries required for expression evaluation, parameter transmission, and space for local data objects declared within the procedure. If an executing procedure calls another, or itself (recursive call to be discussed in Section 9.2.3), a new activation record is placed on the top of the stack (dynamic nesting of activation records). Upon procedure termination, that procedure's activation record is popped off the stack with resumption of the original procedure.

The run-time stack of activation records grows upon entry to an inner level of dynamic nesting and shrinks on exit therefrom.

Since REVS stores all local/shareable data within the dynamic run-time stack, rather than statically at compile time, a significant amount of memory space is saved. In the current REVS design, the maximum number of "different" procedures active at any one time is only 29 (this number does not account for multiple activations of a recursive procedure). The data storage space required if all 1108 REVS Pascal procedures were active at one time would be 209,257 bytes, which is 89K bytes larger than the current stack space allocation of 120K bytes. Dynamic stack allocation of memory space not only reduces the total run-time memory requirements, it also provides a mechanism for supporting recursive subprogram calls (see Section 9.2.3).

The second type of dynamic storage management used in REVS is heap storage allocation. A heap is a block of memory within which pieces can be allocated and deallocated explicitly through Pascal language constructs (NEW and DISPOSE). The built-in function NEW is used to obtain space dynamically during program execution. This space (data structure) is not referenced directly by name, but indirectly by a pointer variable (points to the location containing the value). Heap space allocation is useful for implementing data structures whose size varies as the program is running. For example, linked-lists where the number of elements is unknown at compile-time. Linked lists are extensively used by the RADX Function. Their size is dependent on both the operation to be performed and the size of the application data base. Once this space is no longer needed, the built-in function DISPOSE is used to recover the space for reuse.

REVS implementation of dynamic storage allocation combines the heap and stack within a common block of memory (120K bytes), but starting at opposite ends and growing toward the middle. This feature provides for economical use of dynamic storage space.

FORTRAN Implementation

FORTRAN was designed for strictly static storage allocation. To convert REVS and maintain existing capabilities, the following re-design would be required.

- All data must be allocated storage space statically at compile-time rather than dynamically during execution.
- All variable size data structures would have to be redesigned, either by specifying a maximum fixed size or by using secondary storage such as a disk file. In either case, a major redesign of the affected subprograms/data structures would be required.
- A fixed tradeoff between memory space and run-time performance would have to be made. This would result in unused memory space for small applications and operation-dependent performance degradation for large applications.

COBOL Implementation

The COBOL language does not support dynamic memory allocation. Therefore, the same changes/modifications as were described for a FORTRAN implementation would be required if REVS were converted to COBOL.

JOVIAL J73 Implementation

The JOVIAL J73 language supports dynamic stack allocation but not heap allocation. Therefore, the stack would have to be resized to meet dynamic stack-only requirements since stack and heap space would no longer be obtained from a common block of memory. A scheme for managing a statically defined block of storage space through the use of data structure overlays would have to be designed to replace the current heap management logic. The data structures size (for example, a linked list) will be statically fixed at compile time. Therefore, a tradeoff will have to be made between memory space usage and processing capability.

ADA Implementation

The ADA language supports both stack and heap dynamic management. One difference that currently exists is that ADA doesn't provide a language construct for explicitly disposing/releasing allocated dynamic space. Once dynamic variables are allocated heap space, they remain allocated until the program/subprogram unit containing the access/pointer variable definition (this variable points to the location in the heap containing the data), completes execution. In some ADA implementations, this space may be recovered for reuse through a garbage collection technique. Without a language construct similar to the Pascal DISPOSE construct, the REVS heap management logic will have to be redesigned.

9.2.3 Recursive Subprogram Calls

Pascal permits procedures to invoke themselves, either directly, or indirectly via another procedure. These procedures are said to be recursive. Many algorithms are most naturally represented using recursion; for example, traversing binary trees in the REVS data base. Historically, because of the influence of FORTRAN and COBOL which do not support recursion, recursive algorithms have been neglected for an iterative solution.

Recursive execution of a subprogram requires that the changeable information (local variables, etc.) associated with a subprogram being executed be stored separately for each instance of execution/activation. As was described in the previous section, a dynamically managed run-time stack is utilized within REVS to store a procedure's changeable information/data for each activation.

REVS utilizes recursive logic extensively. The following REVS functional modules have the indicated number of recursive subprograms.

- REVS Executive (12)
- RSL Translation (56)

- Requirements Analysis and Data Extraction (31)
- Interactive R_NET Generation (3)
- Simulation Generation (8)
- CALCOMP Plotting (1).

Of the 1108 Pascal procedures, 111 of them are recursive. Also, the Compiler Writing System, used to generate the RSL Translator, has one recursive procedure.

FORTRAN Implementation

FORTRAN does not support recursive algorithms. In order to support recursion, FORTRAN would require a dynamically created run-time stack of subprogram linkage data (return addresses, etc.). Since FORTRAN is not implemented with dynamic storage management, recursive calls are illegal. Therefore, in order to convert REVS to FORTRAN, the 111 REVS recursive subprograms would have to be redesigned to solve the specific problems non-recursively.

COBOL Implementation

The COBOL language, like FORTRAN, doesn't support recursive algorithms. See discussion under "FORTRAN Implementation" above.

JOVIAL J73/ADA Implementation

Both languages provide for recursive subprogram calls.

9.2.4 Structured Programming Techniques

Structured programming constructs are used to provide code that is easier to read, understand, debug, and maintain. Structured programming, in its most limited definition, consists of a limited number of constructs that specify the flow of control of the program. For example:

- Sequences of two or more operations.
- Conditional branch to one of two operations and return (IF a THEN b ELSE c).
- Repetition of an operation while a condition is true (DO WHILE).

Each of the three structures itself represents a proper program. Using combinations of these basic structures, any program can be built. REVS also includes additional Pascal constructs where necessary to provide more readable and self-documenting programs, more efficient programming, and programmer conveniences.

For example, the CASE statement is used to select one statement (or series of statements) for execution out of a set of statements. The emphasis in REVS is on clear, not clever, programming.

FORTRAN Implementation

FORTRAN 77 does support structured programming, but few compilers are currently available.

FORTRAN IV (1966 standard) does not support structured programming, but structured FORTRAN Preprocessors are available for use.

Therefore, without using FORTRAN 77, or a preprocessor, the conversion of Pascal structured programming constructs into FORTRAN IV equivalent constructs would require heavy use of statement labels and GOTOs which are currently rarely used. A FORTRAN implementation would be a step backwards.

COBOL Implementation

COBOL does provide the basic constructs used in structured programming; except for the CASE statement which could be accomplished by nested IFs.

JOVIAL J73/ADA Implementation

Both of these languages support the structured programming techniques currently used in REVS.

9.2.5 Data Structures

When a software designer uses a higher order language to solve a problem, he must first decide on a way to represent or encode the problem data in terms of the data structures provided by the language. The Pascal data structures utilized within REVS are:

- Simple variables
- Multi-dimensional homogeneous structures
- Multi-dimensional heterogeneous structures

Pascal constructs allow each item of a structure to have a self-documenting descriptive identifier to help avoid misunderstandings about sophisticated complex structures during program life-cycle maintenance.

FORTRAN Implementation

FORTRAN does not support heterogeneous structures. A conversion to FORTRAN would require all such structures to be redesigned in terms of basic homogeneous arrays. Such a step would detract from the code's readability.

COBOL/JOVIAL J73/ADA Implementation

Each of these candidate languages provide the data structures necessary to adequately represent the current REVS design.

9.2.6 Automatic Consistency Checking

Pascal requires that all data items be explicitly declared and associated with a specific data type (integer, real, character, etc.). In addition, a range of values that the specified data type can take on can be specified. For example, an integer variable COUNT is declared to have a sub-range of integer values 0 through 9. During compilation, if an assignment statement is detected setting COUNT to anything other than 0 through 9, it is flagged as an error (attempt to assign an out-of-range value). However, if the value of the "integer type" variable TEST is assigned to COUNT, this statement cannot be analyzed for correctness until run-time (the value of TEST is unknown at compile time). During program execution, Pascal will optionally check all assignment statements for correctness. This run-time consistency checking is performed whenever REVS is run in the Pascal "debug mode" (which is the normal configuration).

None of the candidate languages provide run-time consistency checking, but ADA and JOVIAL J73 do provide compile time type checking. In order to maintain run-time checking (range checking) if REVS were to be converted, inline code would have to be added to perform the checks. This would be expensive in terms of the additional execution time required to perform the test, and the space required to store the extra code. Without range checking, whether implemented automatically by the language compiler or manually by the software designer, a process critical parameter could be assigned a value which, when utilized, could produce unpredictable results. Therefore, a cost effective approach would be to provide run-time consistency checking for only those items identified to be operationally critical.

9.2.7 Input/Output

REVS utilizes both Pascal built-in procedures and FORTRAN routines to perform Input/Output to and from external storage devices (disk files, magnetic tape, cards, and interactive terminal). The Pascal procedures are used to read/write from a device sequentially, while the FORTRAN I/O routines are used whenever random access is required. All the candidate languages provide built-in I/O procedures, with one exception. JOVIAL J73 does not, but does support the calling of FORTRAN I/O routines.

9.2.8 Automatic Generation of RSL Translator

The RSL Translation function, within REVS, provides the mechanism to add, modify, or delete information currently stored in the ASSM. The RSL translator is written in Pascal with relevant portions generated automatically by the Lecarme-Bochmann Compiler Writing System (L-B CWS).

The RSL translator can be thought of as operating in two phases; the analysis of the input source text and the synthesis of the object text. The analysis phase consists of the decomposition of the source text into its basic parts. The synthesis phase consists of the construction of equivalent object program parts from these basic parts. The analysis phase normally builds tables which are used in both analysis and synthesis operations. In terms of RSL translation, the ASSM serves as one large table containing the accumulated information both from this translation execution and prior translation executions.

Conceptually, the translator performs four functions distributed over the analysis and synthesis phases. These functions are lexical analysis, syntax analysis, semantic analysis, and error handling. There is a considerable degree of interaction between these functions. In particular, the syntax analyzer can access any of the other three functions and the error handling function may be accessed by any of the other three functions.

The lexical analyzer or scanner is the simplest part of the translator. Its function is to scan the characters of the source text from left to right and build the actual symbols of the data base. These symbols include identifiers, key words, and single or multiple character punctuation marks. These symbols are variously referred to as lexical or syntactic units, tokens, or atoms. In terms of RSL, these units are divided into punctuation marks, words, numbers, and text strings. The symbols are passed on to the syntax analyzer, usually in the form of integers or other fixed-length symbols rather than variable length strings of characters.

The syntax analyzer or parser performs the more difficult task of determining how the syntactic units received from the lexical analyzer can be grouped together to form the hierarchical structure, called the derivation tree, which indicates how the source text decomposes into the rules of the grammar defining the language.

The semantic analyzer associates a meaning with the derived hierarchical structure. It checks the structure for semantic correctness and stores necessary information about the structure in the symbol table, which for REVS is the ASSM.

This orderly scheme of lexical, syntax, and semantic analysis is adequate, however, only if the source text contains no errors. In practical applications, the source text can be expected to very often contain errors which must be recognized and treated by the translator. These errors can be either lexical, syntactic, or semantic in nature; requiring a general error handling function accessible from these three analyzers.

Lecarme-Bochmann Compiler Writing System

The Lecarme-Bochmann Compiler Writing System (L-B CWS), developed at the University of Montreal, accepts as input an integrated description of a language (RSL) and produces as output a translator for that language.

There are several advantages to use of a syntax-directed CWS:

- It is readily responsive to changes in the design of the language, whether to accommodate changes in user's needs, or in order to achieve internal consistency.
- Use of metalinguistic description as input to the CWS assures that the language intended by the designers is actually implemented.
- A compiler or translator written with a good CWS is vastly simpler to code and debug, therefore reducing maintenance costs.

The RSL Translator's lexical and syntax analyzer functions are currently generated using the Lecarme-Bochmann CWS. Therefore, if REVS were to be converted, the CWS could have to be redesigned in order to produce the RSL translator's lexical and syntax analyzers in the candidate language. Since ADA and JOVIAL J73 are similar to Pascal with respect to the following:

- Block-Structured language
- Strongly typed language
- Recursion

the redesign would not be as extensive as a FORTRAN or COBOL implementation.

9.2.9 Automatic Generation of Simulator Program

The automatic simulator generation capability in REVS takes the ASSM representation of software requirements and generates a discrete event, closed-loop simulator written in Pascal. The simulator functional components are shown in Figure 9-1.

The REVS simulator generator (SIMGEN) transforms the ASSM representation of software requirements into "simulator models" in the Pascal language. Processing flow through the simulated software system is specified in the ASSM as requirements networks, also called R_NETs. Each R_NET identifies an ordered sequence of processing steps (ALPHAs) to be performed by the software system. The REVS simulator generator produces a Pascal procedure for each R_NET to be simulated. Each processing step (ALPHA) referenced on the R_NET becomes a call to a user-supplied model of the process (BETA or GAMMA), stored as an attribute of the ALPHA.

Two distinct types of simulators may be generated by REVS. The first is a simulator which uses "functional models (BETAs)" of the processing steps (ALPHAs) to be performed by the subject system. This type of simulation serves as a means to validate the overall required flow of processing against higher level system requirements. The other type of simulator uses analytic models (GAMMAS), i.e., models that use actual algorithms similar to those which will appear in the actual software to perform complex computations.

These models/algorithms are written by the REVS user in Pascal and RSL statements and entered into the ASSM as textual attributes of the ALPHAs. The RSL statements provide the BETAs and GAMMAS with data file manipulation capabilities which augment the Pascal language. During simulator generation, the RSL statements, identified by scanning the BETA or GAMMA text for RSL keywords, are translated into Pascal statements necessary to accomplish the specified RSL operation. The RSL keywords recognized by SIMGEN are as follows:

- CREATE - The CREATE statement adds a new record to a file.
- SELECT - The SELECT statement is used to make available to the BETA or GAMMA the contents of one record (instance) in a file.

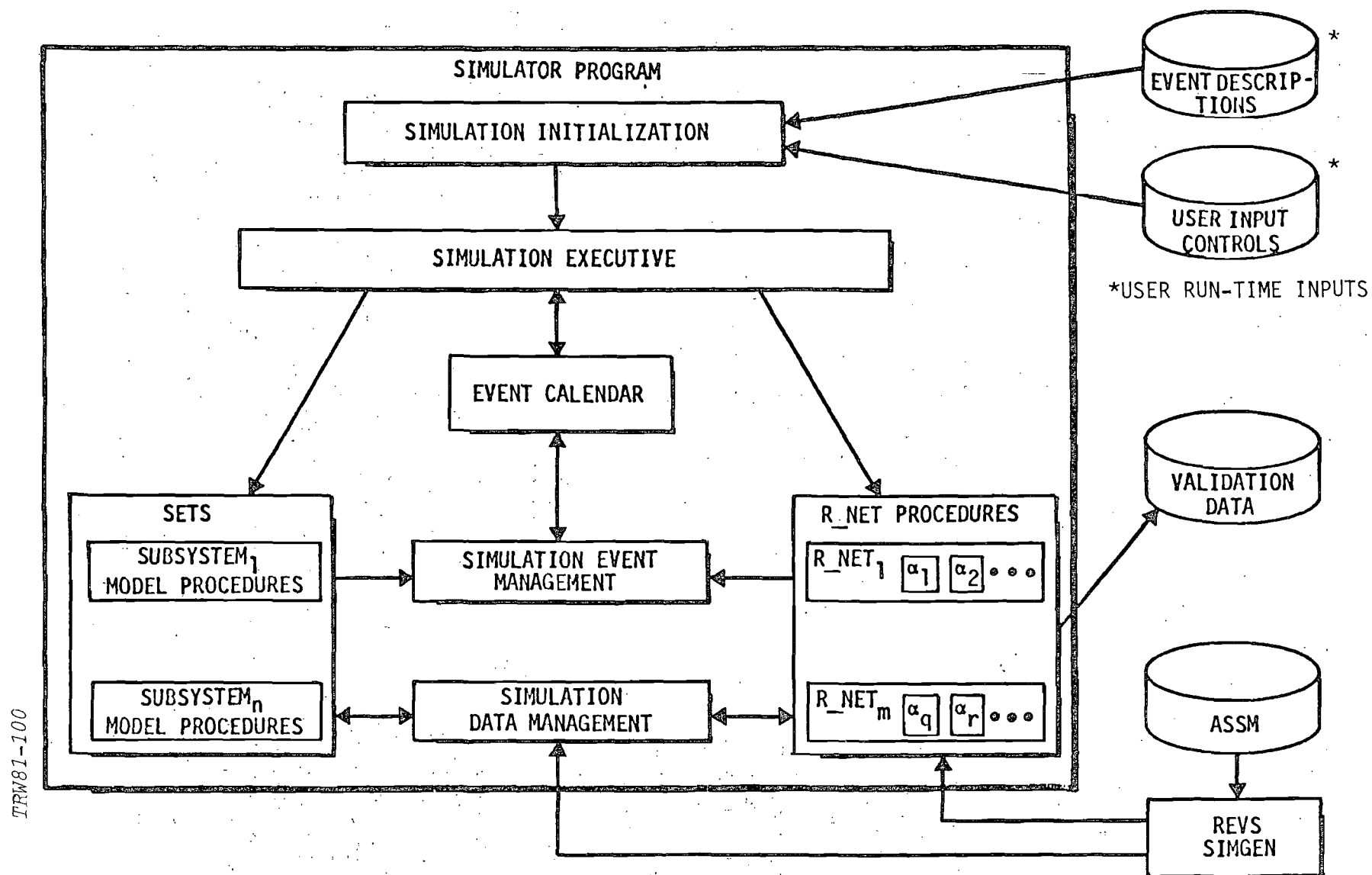


Figure 9-1 Simulator Program Overview

- DESTROY - The DESTROY statement removes the currently selected record from a file.
- FOR EACH - The FOR EACH statement is an iterative form of the SELECT statement. It allows a simple means of applying common operations to multiple records in a file.

These RSL statement keywords, as well as any new keywords which might be defined, must not conflict with the REVS implementation language reserved words.

The data definitions and structures used by the simulator are synthesized from the required data elements, their relationships, and their attributes in the ASSM. The Pascal code generated from the ASSM is automatically combined with externally generated simulation support routines (executive, event calendar, event manager, data managers, etc.), and a System Environment and Threat Simulator (SETS), a generic name assigned to the driver, is developed external to REVS in Pascal. Each external subsystem referenced by the software requirements is modeled within the driver. The simulation support routines are as follows:

- Simulation Executive - The Simulation Executive controls the sequence of execution of R_NET and external subsystem models. The execution sequence is determined by the order of events on the event calendar. Code is generated by SIMGEN such that the required models can be invoked during simulation.
- Event Calendar - The event calendar is a time-ordered linked list of events (execution of R_NET or external subsystem models) to be executed.
- Simulation Event Manager - The Simulation Event Manager provides the utilities necessary to correctly maintain the event calendar.
- Simulation Data Manager - The Simulation Data Manager supports requests for data from both the simulation driver subsystem models and the software requirements models. SIMGEN produces the source code necessary to access the data structures constructed from the ASSM data definitions.

Finally, the various parts (SIMGEN generated code and externally generated code and externally generated code) are compiled and linked producing an executable simulator. A REVS Executive utility (written in host system assembly language) is called to invoke the compiler and linkage editor to generate the load module. Further details of the simulation construction software and the complex stream of job steps invoked are discussed in Paragraph 2.2.4.

If REVS were to be converted to one of the candidate languages, the SIMGEN software would have to be redesigned to produce code in the candidate language. Also, the externally generated components, for example SETS, would have to be developed in the new language. Since ADA and JOVIAL J73 are similar to the current REVS implementation language with respect to the following:

- Block-Structured language
- Strongly typed language
- Support of recursive algorithms

the redesign would not be as extensive as a FORTRAN implementation. A COBOL implementation would not be feasible because COBOL is very limited and awkward with respect to arithmetic capabilities necessary to support simulations for C^3 and weapon system problems. However, the REVS SELECT construct would not be allowed in ADA because it is a reserved keyword with an entirely different ADA meaning. A new RSL keyword, such as CHOOSE, would be necessary, thus altering the current RSL baseline definition. This change would require updates to all user manual, methodology manual, and training manual documentation, as well as changes to the REVS software.

9.2.10 Automatic Generation of Simulation Post-Processor

When generating an analytic simulator (discussed in Section 9.2.9), REVS also automatically generates a simulation post-processor for use in evaluating the performance requirements to be met by the specified software. The ASSM representation of each requirement has an attribute, TEST, which is used by the software specifications engineer to define an executable test module. In order to generate the data/information to be processed by this user-defined TEST module, VALIDATION POINTS are defined at various points in the software requirements networks. Each time during simulation that the processing flow reaches a VALIDATION POINT, data/information are transferred to a recording system which records the relevant information for post-simulation analysis. During post-processing, an executable TEST module can then determine whether accuracy and timing performance requirements have been satisfied by the software.

A TEST is written in standard Pascal and RSL statements and entered into the ASSM as textual attributes of a specific software performance requirement. The RSL statements are used to access validation point data/information, and are translated by SIMGEN into legal Pascal code. Since these special statements are similar to those utilized in BETAs and GAMMAs, their description and conversion considerations are not repeated in this discussion (see Section 9.2.9).

The Pascal code generated for a TEST is automatically combined with externally generated post-processor support routines:

- Simulator Post-Processor Executive
- Simulator Post-Processor Initialization
- Simulator Post-Processor Data Manager

for post-processor execution outside the control of REVS.

The language conversion considerations are the same as those presented in Section 9.2.9 (Automatic Generation of Simulator Program).

9.3 CONVERSION COST/SCHEDULE

The cost associated with the conversion of REVS Pascal code into each of the candidate languages is a function of the following:

- How much existing code can be utilized without change?
- How much existing code must be rewritten because of language syntax and semantic differences?
- How much existing code must be redesigned because the candidate language does not adequately support a REVS feature/capability?

Variations among the candidate languages in syntactic structures (symbolic notation) will necessitate the rewriting of REVS code in the correct symbolic notation. The underlying meaning (semantics) will usually be the same even though the symbolic notation is different. For example, FORTRAN and JOVIAL use the symbol '=', while Pascal and ADA use ':=' to represent an assignment operation. The types of syntax changes required if REVS is converted are as follows:

- Identifiers (length restrictions, type of characters).
- Operator symbols.
- Keywords and reserved words (designate statement type delimiters within statements).
- Comments and noise words.
- Blanks.
- Delimiters.

No matter which language is selected, nearly every one of the 47,530 Pascal statements will have to be rewritten. This task can be simple where a one-to-one correspondence between language constructs exists, rather than an absence of or significant variation of a construct in the target language. Close correspondence makes construction of an automated language-to-language translator a cost-effective approach to conversion. Of the four candidate languages, ADA rates first with respect to syntax similarity with Pascal, followed by JOVIAL, FORTRAN, then COBOL.

When programming in a language that does not provide a desired feature directly, the programmer necessarily must provide his own implementation utilizing the primitive elements provided by the language. In the previous section (9.2), the features and capabilities of REVS which are dependent, either directly or indirectly, upon the current implementation language, Pascal, were presented with a discussion of the adequacy of each candidate language to support those features. Table 9.1 presents a rating (maximum score of forty-four) of how well each candidate language supports the existing features. The lower the rating, the greater the redesign necessary. A zero indicates that the feature is not provided by the language, requiring the design of an alternate solution. The results of the evaluation indicate the lowest amount of redesign if existing REVS code were converted to ADA.

Table 9.1 Candidate Language Comparisons

TRW81-040

REVS FEATURE	CANDIDATE LANGUAGE	ADA	JOVIAL J73	FORTRAN IV (1966 STD)	COBOL
MODULAR ARCHITECTURE		4	4	4	0
STATICALLY NESTED PROCEDURES		4	4	0	0
DYNAMIC STORAGE MANAGEMENT		3	2	0	0
RECURSIVE ALGORITHMS		4	4	0	0
PARAMETER PASSING		4	4	4	0
DATA TYPING		4	4	0	0
INPUT/OUTPUT		4	2	4	4
DATA STRUCTURES		4	4	2	4
STRUCTURED PROGRAMMING CONSTRUCTS		4	4	0	3
LANGUAGE SYNTAX SIMILARITY		3	2	1	0
AUTOMATIC RANGE CHECKING		3	3	0	0
TOTALS		41	37	15	11

This conclusion is supported by an independent study sponsored by the Air Force Avionics Laboratory under Contract F33615-78-C-1466. The results ("DoD's ADA Compared to Present Military Standard HOLs, A Look At New Capabilities") were presented at the 1980 National Avionics and Electronics Conference (NAECOM), by Systems Consultants, Inc. [11]. ADA, rated first, scored 373 points out of a possible 394 points. JOVIAL J73 was second with 290 points followed by FORTRAN at 177. (COBOL was not evaluated in that survey. However, the Navy CMS-2 language, with 210 points, scored ahead of FORTRAN, relegating FORTRAN to last place.)

The kinds of software tools needed to translate REVS Pascal code into one of the candidate languages are as follows:

- Interactive Text Editor.
- Candidate language compiler for the host machine.
- Program to identify all potentially recursive subprograms (available in-house at TRW).
- Program to identify static nesting of subprograms (available in-house at TRW).
- Program to automatically replace specific Pascal symbolic notation with that of the candidate language. This will be possible where a one-to-one mapping between language constructs exists (must be developed).

As an example of automatic translation potential, the Pascal WHILE statement


```
WHILE NUMBER > 0 DO
```

```
BEGIN
```

```
SUM:=Sum + Number;
```

```
NUMBER:=Number - 1;
```

```
END;
```

can be automatically translated into the following ADA statement

```
WHILE NUMBER > 0 LOOP
```

```
SUM:=Sum + Number;
```

```
NUMBER:=Number - 1;
```

```
END LOOP;
```

The estimated effort to accomplish the conversion of 47,530 Pascal statements into each of the candidate languages, develop/modify required documentation, and perform validation testing is as follows:

<u>LANGUAGE</u>	<u>ESTIMATED MANMONTHS</u>	<u>ESTIMATED SCHEDULE (MONTHS)</u>
ADA	108	15
JOVIAL J73	145	18
FORTRAN	280	24
COBOL	394	36

It can be seen that converting REVS to a similar type language like ADA, requiring a minimum amount of redesign, is the most cost-effective approach. These estimates were validated with TRW's Software Cost Estimating Program (SCEP). They represent technical/engineering effort and do not include management/secretarial support costs.

9.4 CONVERSION IMPACT UPON LIFE-CYCLE MAINTENANCE

Software life-cycle maintenance costs are influenced primarily by the following:

- Understanding the existing software. This implies the need for good documentation, good traceability between requirements and code, and well-structured and readable code.
- Modifying the existing software. This implies the need for modular software which minimizes the side effects of changes to code or data structures.
- Revalidating the modified software. This implies the need for software structures which facilitate selective retest, a

standard set of validation tests, and aids for making retest more thorough and efficient.

It would not be cost effective to rewrite REVS in one of the interim HOLs. First, the current Pascal implementation of REVS exists at five sites within the DoD community. Enhancements made at the various sites can now be easily shared by all, reducing the total DoD REVS maintenance cost. Second, since ADA, the proposed DoD standard HOL, is a Pascal-like language, it would be very cost-effective to install the Pascal version in the STE until sufficient reliable ADA compilers are available within the DoD community. This interim period could be utilized by STE personnel to become familiar with the existing features/capabilities, and those enhancements made at other sites for possible inclusion in the eventual ADA version.

A caveat should be noted here about the use of a HOL for requirements specification that is the same as the eventual implementation language. While there are human engineering benefits, in that the user need not be familiar with two languages, there is a greater danger. Namely, that of developing a design, and imposing it as a set of requirements. Description of requirements using Pascal as a base language has the advantage of separating requirements definition from design of the operational software.

9.5 LANGUAGE CONVERSION CONCLUSIONS

The purpose of this study was to evaluate the feasibility of rewriting REVS, currently implemented in Pascal, in one of the DoD approved HOLs. The results of our evaluation indicate that the most cost-effective conversion would be to ADA. JOVIAL J73 would be second choice. In view of the future trends in programming languages over the next ten years, conversion to FORTRAN or COBOL would be a costly step backward. The trend in the micro-processor/personal computer community is toward Pascal instead of the older traditional languages, such as FORTRAN. Because of Pascal's increasing popularity, its similarity to ADA, and since other DoD installations are using the Pascal version of REVS, we recommend a Pascal version for the STE until a proven ADA compiler is available. A J73 implementation is possible, but its use and support would be limited primarily to the Air Force. This would deny the Air Force direct incorporation of REVS improvements made on behalf of other DoD sponsors. A decision on an ADA or J73 implementation should be suspended until the future of both languages becomes more clear.

10.0 REFERENCES

1. "REVS User's Manual - Revision B" TRW Report 27332-6921-026, 28 June 1979.
2. "REVS Maintenance Manual - Revision B" TRW Report 27332-6921-026, 28 June 1979.
3. Richards, K. C. and J. C. Richardson, "PERCAM Post-processor User's Manual", TRW Report 28375-6921-001, November 1975.
4. Callaway, L. S., R. C. McCoy, et al, "Air Defense Systems Performance Analysis Final Report, Volume 2: PERCAM User's Manual", TRW Report 30749-6921-001, 26 October 1977.
5. Lamb, S., et al, "Computer Program Development Specification for IDEF Support Tools (BUILD 1).", Boeing Computer Services Co. Report BCS-40254 (Revised), 1 April 1979.
6. "Computer Program Product Specification for IDEF Support Tools (BUILD 1)", Boeing Computer Services Co. Report BCS-40260, 13 July 1979.
7. Boeing Computer Services Co., "ICAM Computer Program Development Specification for AUTOIDEF₀ (1.5)", 30 May 1980.
8. Shneiderman, B., "Human Factors Experiments in Designing Interactive Systems", IEEE Computer Society Computer Magazine, Vol. 12, No. 12, pp. 9-19 (December 1979).
9. Yates, E. H., "Interrelationships of Technology, System Performance and Prices for Mini/Midicomputers", General Research Corp., (Huntsville, AL) Report TIO 2286, August 1980.
10. Cottrell, J., C. Shu and G. Short, "Multi-Level Security for Intelligence Data Processing Systems", TRW Final Technical Report for Contract F30603-77-C-0119, 30 September 1978.
11. Scheer, L. S., and M. G. McClimens, "DoD's ADA Compared to Present Military Standard HOLs, A Look at New Capabilities", IEEE Reprint CH1554-5-1/80/0000-0539\$00.75.

APPENDIX A

ARPANET HOST HARDWARE/SOFTWARE ENVIRONMENT DATA

ARPANET HOST: Air Force Weapons Laboratory, Kirtland Air Force Base

ACRONYM: AFWL

LIAISONS: NAME

ARPANET ADDRESS

PHONE NO.

Roy Maul

afwl@i4-tenex

(505) 844-2581

MACHINE MAKE/MODEL: CDC Cyber-176, 730, 6600

OPERATING SYSTEM: NOS/BE

PRIMARY STORAGE: 1 megabyte each (Cyber-176 has some "fast disk" ECS)

SECONDARY STORAGE: Many 844s, many 814s

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4014, FR80 and Calcomp support. No color raster support.

COMMUNICATIONS LINES: 10 dial-up, many dedicated lines.

PASCAL COMPILERS: Unknown

FORTRAN COMPILERS: Unknown

STATUS/COMMENTS:

Liaison knows little about AFWL user services and operations. He could recommend no other contacts. Presumably Pascal and FORTRAN compilers are available according to operations personnel at Eglin Air Force Base but their origins and extensions are undetermined.

ARPANET HOST: Argonne National Laboratory, Argonne, Ill.

ACRONYM: ANL

LIAISONS: NAME

ARPANET ADDRESS

PHONE NO:

Lawrence Amiot

amiot@BBN-TENEX

(312) 972-5432

MACHINE MAKE/MODEL: 1 IBM 370/195, 2 IBM 3033's

OPERATING SYSTEM: OS-MVT, OS-MVT-TSO, VM370

PRIMARY STORAGE: 4MB (370) 6MB (3033's)

SECONDARY STORAGE:

NO. DISKS	CAPACITY	MAKE	MODEL
32	317.5 MB	ITEL	7330-12
14	200 MB	ITEL	7330-11
44	100 MB	ITEL	7330-1
14	100 MB	IBM	3330
24	29 MB	IBM	2314

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4014, plans for 4027

Calcomp, Versatek plotters

COMMUNICATIONS LINES: numerous Telnet lines <=1200 baud

PASCAL COMPILERS: not currently supported.

FORTTRAN COMPILERS: more than adequate.

STATUS/COMMENTS:

ANL is willing to upgrade Telnet lines to medium term ("5 year") user's requirements.

Will acquire IBM Pascal as per user requirements.

ARPANET HOST: Brookhaven National Laboratory, Applied Math. Dept.
ACRONYM: BNL
LIAISONS: NAME ARPANET ADDRESS PHONE NO.
Graham Campbell gcampbell@bbn-tenexb (516) 345-4168

MACHINE MAKE/MODEL: 2 CDC 6600s, 1 CDC 7600

OPERATING SYSTEM: SCOPE 3.4

PRIMARY STORAGE: .9 megabytes + 8 megabytes of ECS on 6600a,

.5 megabytes small core + 4 megabytes ECS on 6600b,

.5 megabytes small core + 4 megabytes large core on 7600

SECONDARY STORAGE: 1.44 gigabytes on 6600a, 972 megabytes on 6600b,

2.2 gigabytes on 7600.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4027 and CALCOMP software support.

COMMUNICATIONS LINES: "many dial-up connections, 2 1200 baud lines, 3

Arpanet dedicated lines.

PASCAL COMPILERS: University of Washington.

FORTTRAN COMPILERS: Standard CDC version 4.5.

STATUS/COMMENTS:

ARPANET HOST: Computer Corp. of America, Cambridge, Mass.
ACRONYM: CCA
LIAISONS: NAME ARPANET ADDRESS PHONE NO.
Don Eastlake dee@CCA (617) 491-3670
MACHINE MAKE/MODEL: DEC VAX-11/780
OPERATING SYSTEM: Paging Unix Berkeley version 32

PRIMARY STORAGE: 3.75M
SECONDARY STORAGE: 2 RM03's, 4 300MB drives
GRAPHICS/PLOTTER HADWARE (adequate software support):
3 AED-512 color raster displays. Input: touch, joystick, mouse
Houston Instr. plotter :0" flatbed. Calcomp compatible dial-in/
dial-out
COMMUNICATIONS LINES: 6 dial-up lines, some TELNET lines,
Virtual terminal file-transfer "line" capability

PASCAL COMPILERS: released with Unix tapes.
FORTRAN COMPILERS: released with Unix tapes.

STATUS/COMMENTS:
Not on net yet. Plan to be in near future.
The AED-512 device is comparable to a RAMTEK color raster device
(512x512 pixels, high-speed D operations)
The Unix-released Pascal compiler is a pseudo-code generator,
which is inadequate to RSL/REVS requirements. A version of NBS
Pascal which is partially adequate is being upgraded for release
to Unix environments in the new future.

ARPANET HOST: David Taylor Naval Ship Research and Development Center
ACRONYM: DTNSRDC
LIAISONS: NAME ARPANET ADDRESS PHONE NO.
Robert Tinker dtnsrdc@usc-isie (212) 227-1428
MACHINE MAKE/MODEL: CDC 6400, 6600
OPERATING SYSTEM: NOS/BE

PRIMARY STORAGE: .9 megabytes on each machine
SECONDARY STORAGE: 1.1 gigabytes (6400), 3.3 gigabytes (6600)
GRAPHICS/PLOTTER HARDWARE (adequate software support):
Support for CALCOMP plotters and Tektronix 4027 color raster displays.
COMMUNICATIONS LINES: 4 300 baud TELNET lines,
10 300 baud dialup TTY compatible.

PASCAL COMPILERS: University of Minnesota.
FORTRAN COMPILERS: Standard CDC revision 4.7, level 45.

STATUS/COMMENTS:
DTNSRDC would require a hardware upgrade of its TELNET lines to support the minimum requirement of 4800+ baud communications over the ARPANET for interactive graphics support.

ARPANET HOST: Air Force Armament Division, Eglin Air Force Base

ACRONYM: EGLIN

LIAISONS: NAME

Herbert Spies

ARPANET ADDRESS

spies@bbn-tenex

PHONE NO.

(904) 882-4267

MACHINE MAKE/MODEL: CDC 6600, Cyber 176

OPERATING SYSTEM: NOS/BE

PRIMARY STORAGE: .9 megabytes (6600, 1.8 megabytes (Cyber 176)

SECONDARY STORAGE: 4.14 gigabytes.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4014 and CALCOMP plotter software support. FR80 support.

COMMUNICATIONS LINES: ?????

PASCAL COMPILERS: ?????

FORTRAN COMPILERS: FORTRAN '66, version 4.7, revision level 45.

STATUS/COMMENTS:

ARPANET HOST: Navy Fleet Numerical Oceanography Center.

ACRONYM: FNWC

LIAISONS: NAME

Brian Bradford

ARPANET ADDRESS

fnwc@usc-isie

PHONE NO.

(408) 646-2201

MACHINE MAKE/MODEL: CDC 6500

OPERATING SYSTEM: SCOPE

PRIMARY STORAGE: .9 megabytes + 7 megabytes ECS

SECONDARY STORAGE: 4.8 + gigabytes.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4014 and VERSATEK plotter support. No color raster support.

COMMUNICATIONS LINES: ?????

PASCAL COMPILERS: none.

FORTTRAN COMPILERS: Standard CDC FORTRAN extended to version 4.5.

STATUS/COMMENTS:

Inadequate language and graphics hardware support. FNWC is an ARPANET user and cannot be accessed over the net as a server.

ARPANET HOST: MIT Lincoln Laboratory

ACRONYM: LL

LIAISONS: NAME

Edward Haines

ARPANET ADDRESS

haines@LL

PHONE NO.

(617) 862-5500

x7177

MACHINE MAKE/MODEL: Ahmdah1 470/V7

OPERATING SYSTEM: VM370, VS370

PRIMARY STORAGE: 8 megabytes.

SECONDARY STORAGE: 5.4 gigabytes.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4014, 4015

Ramtek color raster terminals 4-5 of them

COMMUNICATIONS LINES: 8 Telnet lines.

PASCAL COMPILERS: Acquiring IBM Pascal

FORTTRAN COMPILERS: more than adequate.

STATUS/COMMENTS:

ARPANET HOST: Massachusetts Institute of Technology

ACRONYM: MIT-MULTICS

LIAISONS: NAME

ARPANET ADDRESS

PHONE NO:

Richard Scott

scott@MIT-MULTICS

(617) 253-7020

MACHINE MAKE/MODEL: HONEYWELL H-6180

OPERATING SYSTEM: MULTICS

PRIMARY STORAGE: 2.56M

SECONDARY STORAGE: approx. 380M

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektroni 4027

Calcomp (905 device)

COMMUNICATIONS LINES: > >7 for both interactive and FTP

PASCAL COMPILERS: University of Calgary

FORTTRAN COMPILERS: MULTICS-FORTTRAN. Upgraded to '77 next year.

STATUS/COMMENTS:

University of Calgary Pascal, aka PYXIS, may not be adequate to the requirements of RSL/REVS translation.

ARPANET HOST: Naval Air Development Center.

ACRONYM: NADC

LIAISONS: NAME

Ted Calkins

ARPANET ADDRESS

nadc@usc-isie

PHONE NO.

(215) 441-2474

MACHINE MAKE/MODEL: 2 CDC 6600s, Cyber 175, Cyber 760

OPERATING SYSTEM: KRONOS/NOS

PRIMARY STORAGE: .9 megabytes (6600s), 2 megabytes (Cybers)

SECONDARY STORAGE: 9.66 gigabytes for user files.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4027 and CALCOMP support.

COMMUNICATIONS LINES: ???

PASCAL COMPILERS: University of Minnesota versions 2.0 and 3.0.

FORTTRAN COMPILERS: Versions 3 and 4 standard CDC releases.

STATUS/COMMENTS:

The STE has been installed if not fully integrated at NADC,
except for AUTOIDEF.

ARPANET HOST: Naval Ocean Systems Center

ACRONYM: NOSC-CC

LIAISONS: NAME

ARPANET ADDRESS

PHONE NO.

Charles Messinger messinger@NOSC-CC

(714) 225-2168

MACHINE MAKE/MODEL: Univac 1100/82 (250nsec./instr.)

OPERATING SYSTEM: Standard Univac 1100 Operating System

PRIMARY STORAGE: 8 megabytes

SECONDARY STORAGE: 2 8450 drums, 12 8433 disks

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4013,4024

Zeta Plotter 30"max. drum (Calcomp comp. software)

COMMUNICATIONS LINES: 4 comm. ports now, >20 in next 1-1.5 years

PASCAL COMPILERS: Mike Ball's NOSC Pascal

FORTRAN COMPILERS: ASCI FORTRAN level 9 revision 1

STATUS/COMMENTS:

ARPANET HOST: Naval Service Weapons Center, Dahlgren, Virginia

ACRONYM: NSWC-DL

LIAISONS: NAME

Eugene Stemple

ARPANET ADDRESS

nswc-dl@usc-isie

PHONE NO.

(703) 663-8788

MACHINE MAKE/MODEL: CDC 6700

OPERATING SYSTEM: SCOPE 3.4

PRIMARY STORAGE: .9 megabytes

SECONDARY STORAGE: 3.32 gigabytes.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

DISPLA, TEKVIEW support for CALCOMP. Previewing on Tektronix devices.

COMMUNICATIONS LINES: 250Kbyte/sec. channel supporting 16 virtual TTY

TELNET connections or <= 4 FTP connections (1 FTP = 4 TTY virtual connections).

PASCAL COMPILERS: Winograd, University of Colorado.

FORTTRAN COMPILERS: FORTRAN '66, version 4.6, level 433

STATUS/COMMENTS:

ARPANET HOST: Naval Surface Weapons Center, White Oak
ACRONYM: NSWC-WO
LIAISONS: NAME ARPANET ADDRESS PHONE NO.
Robert Archer nswc@usc-isie (202) 394-1909
MACHINE MAKE/MODEL: CDC 6500
OPERATING SYSTEM: NOS
PRIMARY STORAGE: .9 megabytes.
SECONDARY STORAGE: 2.2 gigabytes.
GRAPHICS/PLOTTER HARDWARE (adequate software support):
CALCOMP and Tektronix 4027 support.
COMMUNICATIONS LINES: ?????
PASCAL COMPILERS: University of Massachusetts version 2.0 in user library.
FORTRAN COMPILERS: Standard CDC version 4.7 revision level 45.
STATUS/COMMENTS:

ARPANET HOST: Naval Underwater Systems Center, New London, Conn.

ACRONYM: NUSC

LIAISONS: NAME

ARPANET ADDRESS

PHONE NO.

Don Quigley

dquigley@NUSC-NPT

(203) 447-4349

MACHINE MAKE/MODEL: DEC VAX-11/780

OPERATING SYSTEM: ELF (Arpanet Interface), VMS (host) version 1.6

PRIMARY STORAGE: 1 megabyte

SECONDARY STORAGE: 4 RPO6's for 2 drives

GRAPHICS/PLOTTER HARDWARE (adequate software support)

Tektronix 4014, Calcomp 1055 devices. NUSC is developing in-house device-independent graphics package.

COMMUNICATIONS LINES: 1 9600 baud Server Telnet line.

PASCAL COMPILERS: U. of Washington, i.e., DEC Pascal

FORTTRAN COMPILERS: A good FORTRAN '77

STATUS/COMMENTS:

No FTP support yet. Plans to have it soon via ELF interface.

ARPANET HOST: Naval Weapons Center, China Lake, CA

ACRONYM: NWC

LIAISONS: NAME

ARPANET ADDRESS

PHONE NO.

John Zenor

nwc@USC-ISIE

(714) 939-5559,2

MACHINE MAKE/MODEL: Univac 1100/40

OPERATING SYSTEM: ELF (Imp interface); EXEC-8 (host)

PRIMARY STORAGE: 136K primary, .5M secondary core.

SECONDARY STORAGE: approx. 3G.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Tektronix 4014, 4027 (DISPLA software).

COMP80 microfiche---> plot xerox photo

COMMUNICATIONS LINES: a few supporting 4800b communications.

PASCAL COMPILERS: Mike Ball's NOSC compiler.

FORTRAN COMPILERS: Latest Univac ASCII Fortran.

STATUS/COMMENTS:

ARPANET HOST: Rome Air Development Center, Griffiss AFB, New York
ACRONYM: RADC-MULTICS
LIAISONS: NAME ARPANET ADDRESS PHONE NO.
Robert Walker walker@RADC-MULTICS (315) 330-2501
MACHINE MAKE/MODEL: HONEYWELL 6180
OPERATING SYSTEM: MULTICS

PRIMARY STORAGE: 3 megabytes.
SECONDARY STORAGE: 912 megabytes.
GRAPHICS/PLOTTER HARDWARE (adequate software support):
2-10 Tektronix 4014's, 1 Intecolor color raster terminal (keyed input).
Zeta 36" drum plotter (3 pen holder, micro proc. controller, off-line)
COMMUNICATIONS LINES: 20 entries in virtual interactive terminal line table,
8 FTP ports.

PASCAL COMPILERS: University of Calgary, PYXIS.
FORTRAN COMPILERS: Close to ANSI. Sufficiently robust

STATUS/COMMENTS:

University of Calgary "Pascal" which HONEYWELL Plans to upgrade, release,
and support, especially in MULTICS environments is the only accessible
compiler but it may not be adequate to RSL/REVS requirements.
The Intecolor device has a micro processor controller with dual floppy
disks and will support 9600 baud communication.

ARPANET HOST: University of California at Los Angeles
ACRONYM: UCLA-CCN
LIAISONS: NAME ARPANET ADDRESS PHONE NO.
Robert Braden braden@UCLA-CCN (213) 825-7518
MACHINE MAKE/MODEL: IBM 370/3033 6 mip machine
OPERATING STORAGE: OS/MVT release 21.8, under VM release 6.

PRIMARY STORAGE: 12 megabytes.

SECONDARY STORAGE: 319.5 gigabytes.

GRAPHICS/PLOTTER HARDWARE (adequate software support):

Ramtek 8100 raster

Acquiring IBM color raster hardware

CALCOMP

COMMUNICATIONS LINES: "100 dialup ASCII ports, 6 leased-line ports (two of which are high-speed), 6 NETRJS virtual terminal ports for ARPANET.

PASCAL COMPILERS: IBM Pascal, Hitachi Pascal, Waterloo Pascal.

FORTRAN COMPILERS: Standard IBM release.

STATUS/COMMENTS:

Some question whether FORTRAN compiler supports MASK, SHIFT, ENCODE and DECODE.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

